University of Texas at El Paso
ScholarWorks@UTEP

Open Access Theses & Dissertations

2019-01-01

GPCR-PEnDB: A Database Of Protein Sequences And Derived Features To Facilitate Prediction And Classification Of G Protein-Coupled Receptors

Khodeza Begum
University of Texas at El Paso

Follow this and additional works at: https://digitalcommons.utep.edu/open_etd

 Part of the Bioinformatics Commons

Recommended Citation

Begum, Khodeza, "GPCR-PEnDB: A Database Of Protein Sequences And Derived Features To Facilitate Prediction And Classification Of G Protein-Coupled Receptors" (2019). *Open Access Theses & Dissertations*. 2829.
https://digitalcommons.utep.edu/open_etd/2829

This is brought to you for free and open access by ScholarWorks@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.

GPCR-PEnDB: A DATABASE OF PROTEIN SEQUENCES AND DERIVED FEATURES TO FACILITATE PREDICTION AND CLASSIFICATION OF G PROTEIN-COUPLED RECEPTORS

KHODEZA BEGUM

Doctoral Program in Computational Science

APPROVED:

Ming-Ying Leung, Ph.D., Chair

Lin Li, Ph.D.

Sourav Roy, Ph.D.

Xiaogang Su, Ph.D.

Charlotte M. Vines, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

Copyright ©

by

Khodeza Begum

2019

Dedication

I dedicate this work to my late father, Md. Khorshed Alam and my late sister, Khurshida Jahan who always inspired and believed in me.

GPCR-PEnDB: A DATABASE OF PROTEIN SEQUENCES AND DERIVED FEATURES TO FACILITATE PREDICTION AND CLASSIFICATION OF G PROTEIN-COUPLED RECEPTOR

by

KHODEZA BEGUM, M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Computational Science Program

THE UNIVERSITY OF TEXAS AT EL PASO

December 2019

Acknowledgements

I would first like to thank my advisor Dr. Ming-Ying Leung for her continuous support and encouragement in my research. Her guidance and immense knowledge have always helped me to give my best effort so that I can be successful. It has been an honor to work for her and I would also like to thank the rest of my thesis committee: Dr. Charlotte M. Vines, Dr. Xiaogang Su, Dr. Lin Li and Dr. Sourav Roy for their encouragement, insightful comments and questions.

I thank my Husband, Richard E. Mitchell for his tremendous love and care by always being there for me whenever I needed him the most, also my son, Austin Mitchell, for his unconditional love and encouragement for working harder with his sweet words.

I would like to thank all my amazing friends Fariya Rahman, Himadri Shekhar, Kazi Nusrat Jahan, Marjana Khalil, Nazia Rahman, Sara Tasmeen, Sarahat Afroz and Zarin Tasneem for their continuous support and love from thousand miles away. I am grateful to Sharmin Abdullah for always being there for me and treating me as family. Also, I am thankful to all the wonderful people that I have met at UTEP.

In conclusion, my sincere thanks to Dr. Jonathon E Mohl for always helping me with everything and encouraging me to do better with my work. I am thankful to the Computational Science Program and the NIH Grant #5U54MD007592 from the National Center for Research Resources to the UTEP Border Biomedical Research Center as well as to Gerardo Cardenas for the help in utilizing the bioinformatics computing facilities.

Abstract

G protein-coupled receptors (GPCRs) constitute the largest group of membrane receptor proteins in eukaryotes. Due to their significant roles in various physiological processes such as vision, smell, and inflammation, GPCRs are the targets of many prescription drugs. However, the functional and sequence diversity of GPCRs has kept their prediction and classification based on amino acid sequence data as a challenging bioinformatics problem. There are existing computational approaches, mainly using machine learning and statistical methods, to predict and classify GPCRs based on amino acid sequence and sequence derived features. In this project, we have constructed a searchable MySQL database, named GPCR-PEnDB (GPCR Prediction Ensemble Database), of confirmed GPCRs and non-GPCRs with the goal of allowing users to conveniently access useful information of GPCRs in a wide range of organisms and to compile reliable training and testing datasets for different combinations of computational tools.

GPCR-PEnDB currently contains 3129 confirmed GPCR and 3575 non-GPCR sequences collected from the UniProtKB/Swiss-Prot protein database, encompassing over 1200 species. The non-GPCR entries include transmembrane proteins for evaluating various prediction programs' abilities to distinguish GPCRs from other transmembrane proteins. Each protein is linked to information about its source organism, classification, sequence lengths and composition, and other derived sequence features. Compared to GPCRdb, which is considered the most comprehensive GPCR resource available, our database contains much fewer GPCR sequences because of our requirement for every GPCR to be confirmed. Nevertheless, our database contains 1094 GPCRs not found in GPCRdb. In particular, all of the class D and E GPCRs and many of Class A sensory receptors are missing from GPCRdb.

I will present several examples of using this GPCR-PEnDB along with its graphical user interface to query for GPCRs with specific sequence properties and to compare the prediction accuracies of GPCR prediction tools. This initial version of GPCR-PEnDB will provide a framework for future extensions to include additional sequence features, three-dimensional structural data, and ligand binding information to facilitate the design and assessment of GPCR prediction and classification tools as well as experimental studies to help understand the functional roles of various types of GPCRs.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Biological significance of G protein-coupled receptors

G protein-coupled receptors (GPCRs) are immensely important in numerous physiological processes and are targeted by many therapeutic developments due to their significant roles in intracellular signaling and clinical importance in many diseases, such as cancer, infection, and inflammation [Jo & Jung, 2016]. In particular, they can affect tumorigenesis and tumor growth [Liu et al., 2016]. Also, some of the GPCRs play a vital role in case of heart failure [Wang et al., 2018]. As GPCRs are involved in a wide range of physiological processes including vision, taste, smell, inflammation, cell recognition, pheromone signaling and many more, these transmembrane receptors are being studied extensively by many research groups to understand their biological functions and molecular pathways.

1.2 Prediction and classification of GPCRs

With the advancement of sequencing technologies in the past two decades, large amounts of protein sequence data have become available. Computational approaches have been developed to utilize these sequence data to predict whether a given protein is a GPCR, and if so, classify the functional type it belongs to. Using such computational tools to screen for possible GPCR sequences and make initial predictions of their biological roles prior to detailed experimental investigations can save time and valuable wet-lab resources. In developing new algorithms for GPCR prediction and classification, having a comprehensive and reliable dataset consisting of both positive and negative examples of GPCRs for training and testing is highly beneficial. My dissertation work is motivated by such needs.

GPCRs are also known as 7-TM proteins [Hu et. al, 2017] where every GPCR protein has a characteristic structure consisting of an extracellular N-terminus, an intracellular C-terminus, and between them seven hydrophobic transmembrane helices that are linked through three intracellular and three extracellular loops [Schiöth & Fredriksson, 2005] as shown in Figure 1.

Based on this characteristic structure, many different bioinformatics software tools have been developed for predicting GPCRs and subsequently classifying them hierarchically to gain insights into possible biological functions [Davies et al., 2007]. It is relatively easy to distinguish GPCRs from other non-transmembrane proteins but picking out a novel GPCR from a set of transmembrane proteins may not always be a straightforward task. There are other proteins such as adiponectin receptors that also contain seven transmembrane helices [Hsieh et al., 2005] but differ from the GPCRs in that they have extracellular C-termini and intracellular N-termini, as well as different signaling mechanisms [Alexander et al. 2015]. Other examples of seven transmembrane non-GPCRs include bacteriorhodopsins, rhodopsins from eubacterial and eukaryotic organisms [Hirai et al., 2009] that show similarities with GPCRs.

Figure 1.1: G-protein-coupled receptor with ligand [C. Vines, Personal Communication, December 2017]

After predicting the GPCRs from non-GPCRs, classifying of GPCRs can be based on several features, such as protein sequence homology or functional similarity [Horn et al., 2003], vertebrates [Schiöth & Fredriksson, 2005] or vertebrates and invertebrates both [Alexander et al., 2015]. Horn et al. (2003), divides GPCRs into six major classes, named Class A (rhodopsin-like) [Alexander et al., 2015], Class B (secretin receptor family) [Poyner & Hay, 2012], Class C (metabotropic glutamate) [Palczewski et al., 2000], Class D (fungal mating pheromone receptors) [Eilers et al., 2005s], Class E (cyclic AMP receptors) [Raisley et al., 2004] and Class F (frizzled/smoothened) [(Gentry et al., 2015)]. Among these classes, Class D and E are unique to

invertebrates. GRAFS [Schiöth & Fredriksson, 2005] is a classification system, which divides vertebrates GPCRs into five major classes named as Glutamate [Chaudhari et al., 2000], Rhodopsin-like [Zamanian et al., 2011], Adhesion [Langenhan et al., 2013], Frizzled/Taste2 [Krishnan et al., 2012], and Secretin family [Schiöth & Fredriksson, 2005]. Classifying the major six classes of GPCRs is also known as the secondary level of classification [Davies et al. 2007]. The classes can be furthermore divided into sub-families, sub-sub-families and subtypes as done in GPCRdb [Munk et al., 2016], which is generally recognized as a comprehensive public GPCR resource that has served the scientific community for over 20 years, to indicate the functional types of each GPCR in their collection.

There are several existing computational methods for GPCR prediction and classification [Pearson & Lipman, 1988; Altschul et al., 1990; Elrod & Chou, 2002; Karchin et al., 2002; Qian et al., 2003; Inoue et al., 2004; Guo et al., 2006; Iqbal et al., 2016; Huang et al., 2004; Gao & Wang, 2006; Peng et al., 2010; Davies et al., 2008; Cobanoglu et al., 2011; Li et al., 2010; Sahin et al., 2014; Guerrero et al., 2016; Munoz et al., 2017] . These will be reviewed in Chapter 2. Generally, these methods are trained using GPCR sequences deposited in GPCRdb as positive examples and randomly selected non-GPCRs as negative examples. Their reported prediction accuracies are around the 80 – 90+% range. However, in attempting to evaluate the performance of the different methods and understand their strengths and weaknesses, we came to realize that it is necessary to develop a unified GPCR database that can be used for training and testing different computational approaches or tools due to several reasons:

1. GPCRdb contains a mix of confirmed GPCRs and putative GPCRs that are predicted but not confirmed (see Appendix M). For training and testing puposes, we need a set of confirmed GPCRs, which can be collected from the reviewed GPCRs in the Swiss-Prot database [UniProt Consortium, 2008].
2. The vast majority of the GPCRs deposited in GPCRdb are from human or other mammalian model organisms (Appendix M) and only the non-olfactory GPCRs are included [Pándy-

Szekeres et al., 2018]. Examples from other organisms, such as insects and other arthropods, are not as abundant (Appendix M).

3. The available GPCR prediction web-servers have significantly high false positive rates when tested on the transmembrane non-GPCRs. This issue is discussed in detail in Chapter 6. We, therefore, need a good collection of transmembrane non-GPCRs to be used as negative examples.

1.3 SPECIFIC AIMS

My work focuses on the design and implementation of a searchable database containing confirmed GPCR and non-GPCR sequences to facilitate the development and assessment of different computational approaches and software tools for GPCR prediction and classification. As part of the GPCR-PEn (GPCR Prediction Ensemble) project in our group, this database is named GPCR-PEnDB and is publicly accessible via the Internet through a user-friendly graphical user interface.

Specific Aim 1

Compile a collection of confirmed GPCRs and non-GPCRs from a diverse set of organisms with the aim to provide reliable training and testing data for evaluating various algorithms or software tools developed for GPCR prediction and classification. The dataset will satisfy the following requirements:

1. The dataset should contain both positive and negative examples of GPCRs.
2. There should be proteins from many different species in the dataset.
3. Positive examples should comprise confirmed GPCRs only.
4. Negative examples should span a large variety of proteins, including non-GPCR transmembrane proteins, with different structures and functions.

Specific Aim 2

Develop a searchable database containing the GPCR and non-GPCR proteins in the above dataset, which can be queried using MySQL. This database will not only provide the amino acid sequence data, but also useful information about these proteins (e.g., length, source organism, amino acid composition, etc.) as well as links to their records in UniProt. In addition, a graphical user interface will be developed using the Web.py framework so that users who are not necessarily familiar with the MySQL can access all information conveniently. The database along with the user interface will be made available to the public as part of our group's GPCR-PEn and called GPCR-PEnDB.

Specific Aim 3

Use the GPCR and non-GPCR data collected in GPCR-PEnDB to assess the performance of several existing web-based GPCR prediction tools. Aside from the usual statistical measures of prediction accuracy, special attention will be paid to their capabilities in distinguishing GPCRs from non-GPCR transmembrane proteins.

1.4 Overview of Dissertation

To facilitate the development and testing of computational algorithms and bioinformatics tools for GPCR analyses, including a good collection of negative examples (i.e., non-GPCRs) in the database is just as important as having a diverse set of confirmed GPCRs. This dissertation focuses on the design of the GPCR-PEnDB database and its information can be accessed. Chapter 2 is a review of the existing prediction methods, databases and computational tools used on specific datasets. Chapter 3 describes our process of data acquisition and the resulting datasets. Chapter 4 details the methods we used for creating the relational database and the publicly accessible webserver. This is followed by Chapter 5 that showcases some features in the resulting database and the webserver. In Chapter 6, we present an assessment on the several available webservers

using the protein collection in GPCR-PEnDB as the testing dataset, provide a comparison of their performance, and discuss their strengths and weaknesses. Finally, Chapter 7 concludes with our overall findings and suggests several possible future extensions of the current GPCR-PEnDB.

Chapter 2: Literature Review

This chapter gives an overview of the existing GPCR databases in the public domain, and the different approaches and algorithms used by the researchers to predict and classify GPCRs into various levels. The first section introduces the main GPCR families. The second section describes the available databases and webservers that provide data and information on GPCRs based on their several properties and classification. This is followed by two sections on the algorithms behind the different computational tools used to predict and classify GPCRs and how combinations of two or more algorithms are applied. The final section presents a summary table to show the accuracy levels achieved by the different prediction methods.

2.1 GPCRs

G protein (guanine nucleotide-binding protein) – coupled receptors (GPCRs) are a vast and most diverse group of membrane receptors in eukaryotes. All the GPCRs share a similar formation (Figure 2.1) with an extracellular N-terminus followed by seven transmembrane helices that are connected through three extracellular and three intracellular loops which leads to the intracellular C-terminus [Schiöth & Fredriksson, 2005]. But for certain members of Class A [Sensoy and Weinstein 2015] and Class C [Bruno et al. 2012] GPCRs, an amphipathic helix 8 is found adjacent to the C-terminus that lies parallelly with the membrane surface.

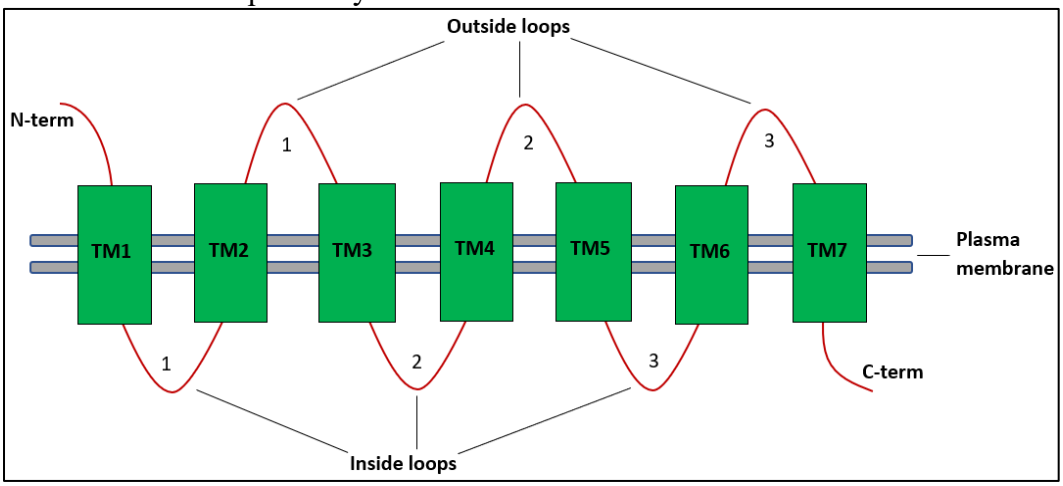


Figure 2.1: The common architecture of a G protein-coupled receptor

Class A

Class A is also known as Rhodopsin-like receptors which is the largest protein family of G-protein coupled receptors [Hu et al., 2017]. The ligands for these receptors comprise of a large number of small molecules, peptides, hormones and neurotransmitters, whereas the receptors include olfactory receptors, taste receptors and five pheromone receptors [Alexander et al., 2015]. The types of receptors that respond to the sensory and the endogenous (non-sensory) signals can also be referred to as chemosensory GPCRs (csGPCRs) and endoGPCRs [Vassilatis et al. 2003] respectively. Vassilatis et al., (2003) also describes how these endoGPCRs (from all the families) are involved in a large variety of physiological processes that regulate metabolism, development, reproduction and so on. There are around 350 endoGPCRs found in human that are the targets of the therapeutic drugs and this is why Rhodopsin-like receptors are studied in a wide range [Eilers et al., 2005] due to their major involvement in prescribed drugs [Venkatakrishnan et al., 2016].

Class B1 or Secretin family

Class B1 are called Secretin receptors which are known for controlling peptide hormones from the glucagon hormone family [Alexander et al. 2015] that is very important in controlling the glucose level of blood or the blood pressure [de Graaf et al. 2017]. Each member of this class has unique properties and plays an important role in metabolic diseases and physiology [Poyner and Hay 2012].

Class B2 or Adhesion family

This family belongs to the GPCR family classified by the GRAFS [Schiöth and Fredriksson 2005] system for classification. Phylogenetically this family has close relationship with the class B (Secretin receptors) family but the receptors have high conservation, larger extracellular N-terminus, cell to cell and cell to matrix adhesion along with unique autocatalytic process which differs from the class B receptors [Hamann et al. 2015]. In GPCRdb Class B1 and Class B2 are

usually combined together for representing Class B and as “G protein-coupled receptor family 2” in Swiss-Prot [UniProt Consortium 2008] from which two of these groups can be separated if needed.

Class C

This family is called the Glutamate family that consists of the metabotropic glutamate receptors along with several other receptors such as calcium-sensing receptor, taste receptors, and calcium-sensing receptors which play a significant role in several physiological processes such as calcium homeostasis, taste and synaptic transmission [Wu et al., 2014]. The unique feature of this family is, it has a large extracellular N-terminus domain which mediates the homodimerization or heterodimerization[Zhang et al., 2014].

Class D

This is a distinct type of family of GPCR which is also known as fungal mating pheromone receptors [Alexander et al., 2015]. This family comprises of a diverse group of receptors and the sequence similarity between the subfamilies is very low [Hu et al., 2017]. The small peptide ligands bind to the extracellular loops and at the ends of transmembrane helices [Eilers et al. 2005].

Class E

Class E is known as the cyclic adenosine monophosphate (cAMP) receptor family of GPCRs, which play a significant role in regulating the development related regulation genes [Xue et al., 2008]. It is also used as intracellular signal transducer and acts as a second messenger for triggering the physiological changes [Raisley et al. 2004].

Class F

This class contains frizzled and smoothened proteins, which is why this family is also known as Frizzled/Smoothened family [Alexander et al., 2015]. The frizzled proteins are regulated by lipoglycoproteins of the WNT family and Smoothened family are indirectly regulated by the Hedgehog family [Alexander et al. 2015].

Taste 2 receptors

This family of receptors are included in the GPCRdb [Pándy-Szekeres et al. 2018] database as a separate family which has a distant relationship with the class A's but the ligand information is not known hence it is classified within the Orphan and other 7TM receptors according to IUPHAR [Alexander et al. 2017] as well.

2.2 GPCR DATABASE AND WEBSERVERS

There are several databases and web-based browsers available that are publicly accessible containing fundamental information on GPCRs as well as structural and signaling pathways. Each database is unique from one another and is briefly described in the following section.

UniprotKB/Swiss-Prot

This database [UniProt Consortium, 2008] contains the information about protein sequences along with the annotation data which is generally updated in every month. It gives overall information about a protein. The main tools provided by this database are BLAST, sequence alignment, ID mapping and peptide search. It creates a collaborative search among four databases namely UniProtKB (protein knowledgebase), UniRef (Sequence clusters), UniParc (Sequence archive) and Proteomes. Currently this database has reviewed 560,537 sequences and 167,761,270 unreviewed sequences which are available through TrEMBL. UniProt is not limited

to GPCR related proteins, but we can search for GPCR sequences and their information in this database.

GPCRDB

GPCRDB contains data, diagram and tools and [Isberg et al., 2016] has the largest collection of receptor mutants and user friendly search option for the collection of crystal structures. Users can create and collect the diagrams like snake-plots, box diagrams and phylogenetic trees for the illustration of receptor residues. There are many features that are unique and this database gives a lot of information such as the sequence alignments considers helix bulges and constriction along with the amino acid conservation statistics with generic residue numbering. This database releases and updates data bi-monthly and currently they have 15,147 proteins, among which 420 are non-olfactory human proteins with a total of 3,547 organisms. It has search options for users such as receptors, signal proteins, ligands with pharmacophore generation and schematic alignments, experimental, mutation browser for crystal structures and provides classification information up through the fifth level.

SEVENS

The SEVENS database [Ikeda et. al, 2018] uses several bioinformatics tools that predicts transmembrane helices, aligns sequences and finds motifs and genes for using it in sixty-eight eukaryotic genomes to accumulate information about the genes in G-protein coupled receptors with four different threshold levels labeled as Level A, B, C and D. Level A and Level D are expected to provide best specificity and best sensitivity respectively by searching for sequence similarity followed by finding motif and domain assignments. The information on GPCRs can be done using several options such as gene/protein lengths, PROSITE motifs, Pfam domains, novelty, family etc., that leads to the detailed analysis of the gene list found for each query.

GLASS

GLASS [Chan et al., 2015] contains the experimentally confirmed information on GPCRs, ligands and their association. This database works with the confirmed GPCRs collected UniProtKB and performs statistical analysis by observing Ki, Kd, IC50 and EC50 with a threshold value to remove the less probable association of GPCRs and ligands. Then with the refined data it cross checks with the available related literature of ligand binding and chemical identifiers. The database contains 3056 confirmed GPCRs and 342,539 ligands which results in a large number of unique GPCR-ligand entries. This database contains the highest number of GPCR-ligand binding information than the current databases.

gpDB

gpDB [Theodoropoulou et al., 2008] is a relational database that provides information on the classes, families and types about 391 G-proteins, 2738 G-protein coupled receptors and how these molecules interact with 1390 with different families and types of effectors. This database provides brief description and cross references on the classes, families, sub-families and sub-types for the available proteins as well as incorporates several tools allowing users perform alignments, BLAST search or run predictive algorithms such as HMMTM, PRED-GPCR, PRED-COUPLE2 and TMRPres2D.

GPCRpred

GPCRpred [Bhasin & Raghava, 2004] is an SVM based method for the prediction of proteins in families and subfamilies level based on the dipeptide composition. As the webserver uses an old classification system, the family level classification is different than the one we have now. In the webserver there are three input options available for the query sequence such as

inserting the standard format (PIR/FASTA/EMBL etc.) or the amino acid single letter code format or upload a file of the sequence.

The output page provides the information of name, sequence, length, date of prediction, the prediction approach (protein feature) used to predict and classify the query sequence. It is observed that the webserver works with one query sequence at a time. Following shows the output page for the query sequence that has been used for the demonstration purpose.

Figure 2.2: GPCRpred webserver with a query sequence

Figure 2.3: GPCRpred output page for the query sequence

PRED-GPCR

This system is based on a probabilistic method [Papasaikas et al., 2004] which uses family specific profile HMMs to determine the classification of GPCRs into families for a given query sequence. The output of the query sequence results in a ranked list of the profile HMM, E-value cutoff, family, P-values, E-values and the number of profiles matched for each family.

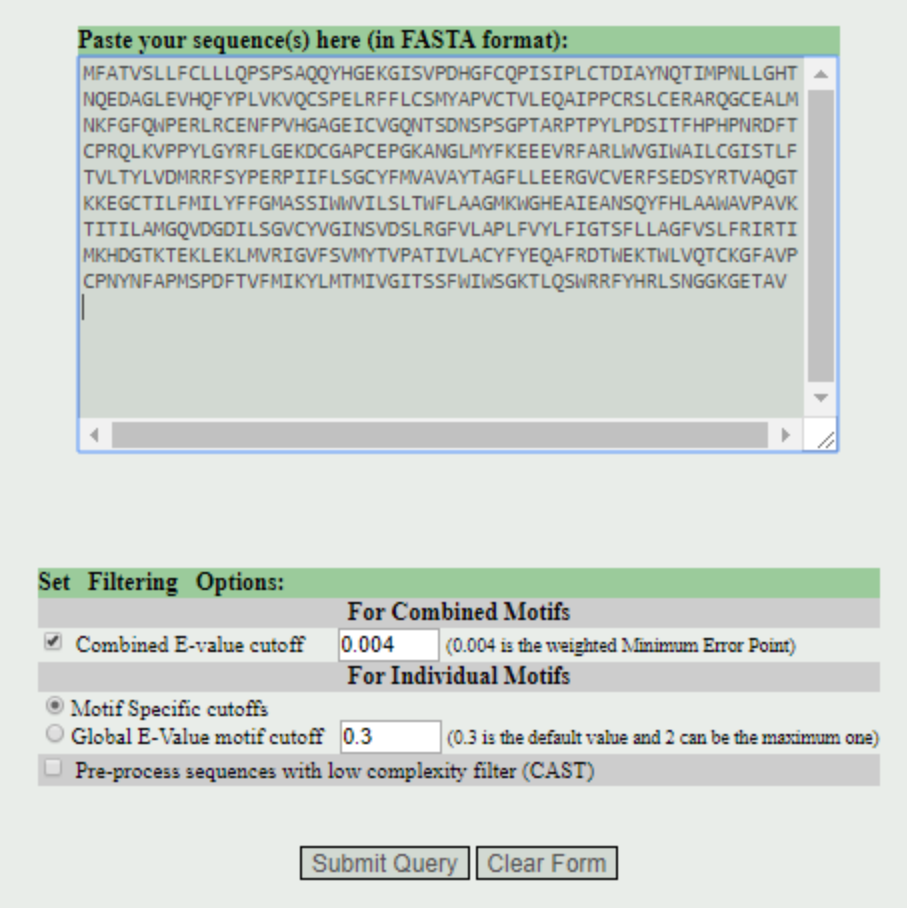
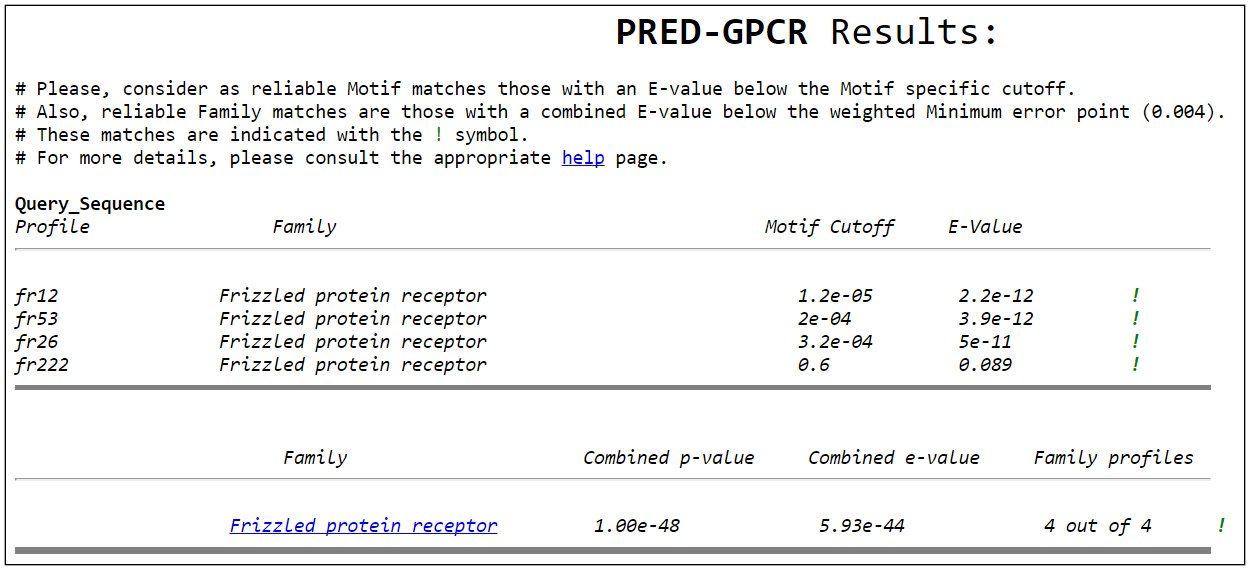


Figure 2.4: PRED-GPCR webserver with a query sequence

After submitting the query sequence, the output result page shows the motif cutoff, e-value, combined p-value, combined e-value and the protein profile information. If a non-GPCR sequence is used as a query, then the result page is blank.



PRED-GPCR Results:

Please, consider as reliable Motif matches those with an E-value below the Motif specific cutoff.
Also, reliable Family matches are those with a combined E-value below the weighted Minimum error point (0.004).
These matches are indicated with the ! symbol.
For more details, please consult the appropriate help page.

Query_Sequence

Profile	*Family*	*Motif Cutoff*	*E-Value*	
fr12	*Frizzled protein receptor*	*1.2e-05*	*2.2e-12*	*!*
fr53	*Frizzled protein receptor*	*2e-04*	*3.9e-12*	*!*
fr26	*Frizzled protein receptor*	*3.2e-04*	*5e-11*	*!*
fr222	*Frizzled protein receptor*	*0.6*	*0.089*	*!*

Family	*Combined p-value*	*Combined e-value*	*Family profiles*	
Frizzled protein receptor	*1.00e-48*	*5.93e-44*	*4 out of 4*	*!*

Figure 2.5: PRED-GPCR output page for the query sequence

SVM-Prot

This webserver [Li et al., 2016] which predicts functional families from the sequences of different types of proteins disregarding the sequence similarity. It provides an option to run a sample sequence available in the server also allows user to choose among three machine learning algorithms such as support vector machine, probabilistic neural network or *k*-nearest neighbor. User can provide multiple sequences to be analyzed, allows to access the result later by providing an email or a link. It also has a feature where users can perform a BLAST for the provided query sequence.

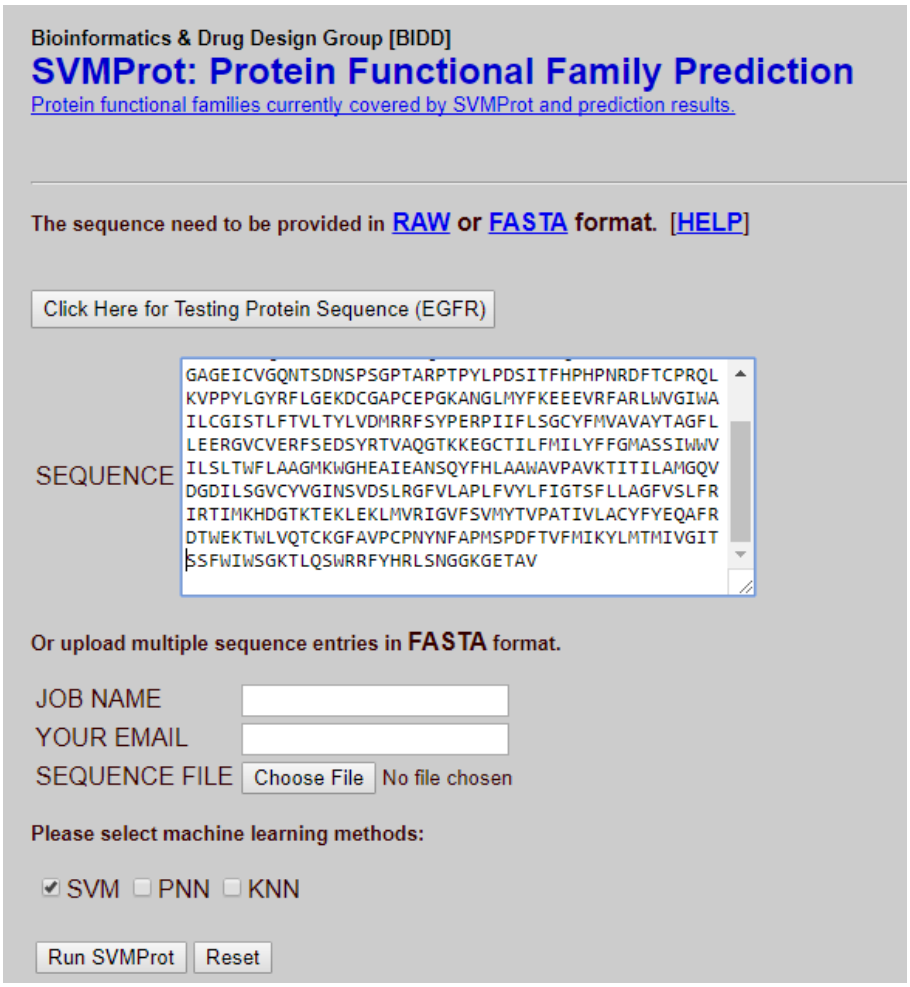
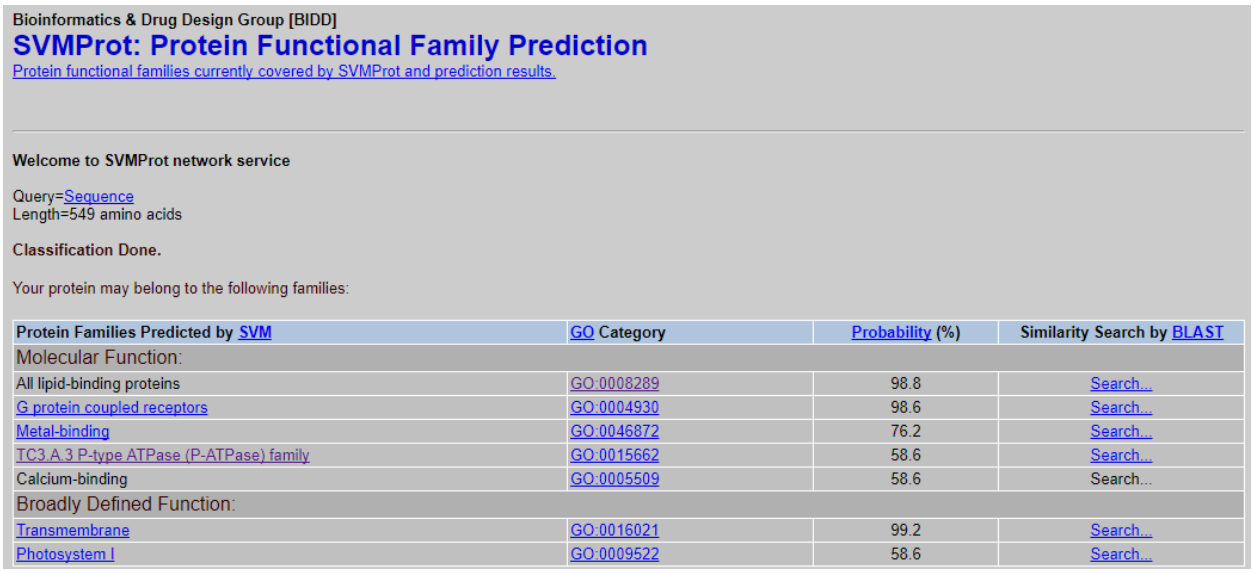


Figure 2.6: SVMProt webserver with a query sequence

The output page initially provides the waiting time and a link to check the results later for the user and when the analysis is done it provides user the result page that shows the probabilities of the protein families the sequence might belong to and the links to access those families with a BLAST option. The following is an example output page for a query sequence.



Bioinformatics & Drug Design Group [BIDD]

SVMProt: Protein Functional Family Prediction

Protein functional families currently covered by SVMProt and prediction results.

Welcome to SVMProt network service

Query=Sequence
Length=549 amino acids

Classification Done.

Your protein may belong to the following families:

Protein Families Predicted by SVM	GO Category	Probability (%)	Similarity Search by BLAST
Molecular Function:			
All lipid-binding proteins	GO:0008289	98.8	Search...
G protein coupled receptors	GO:0004930	98.6	Search...
Metal-binding	GO:0046872	76.2	Search...
TC3.A.3 P-type ATPase (P-ATPase) family	GO:0015662	58.6	Search...
Calcium-binding	GO:0005509	58.6	Search...
Broadly Defined Function:			
Transmembrane	GO:0016021	99.2	Search...
Photosystem I	GO:0009522	58.6	Search...

Figure 2.7: SVMProt output page for the query sequence

GPCR-CA

GPCR-CA [Xiao et al., 2009; Chou, 2001; Chou & Elrod, 1999] predicts and classifies GPCRs into families using a cellular automaton image approach. GPCR-CA first predicts the protein to be a GPCR or non-GPCR, if it is a non-GPCR then the output page result shows that the query sequence is a non-GPCR otherwise for a predicted GPCR, it automates the process to classify the query sequence based on six main functional groups such as Rhodopsin-like, Secretin-like, Metabotrophic/Glutamate/Pheromone, Fungal pheromone, cAMP receptor and Frizzled/smoothened family.

Figure 2.8: GPCR-CA webserver with a query sequence

The output for a query sequence is the length and the family level classification of a protein.

Following is the output result for a query sequence.

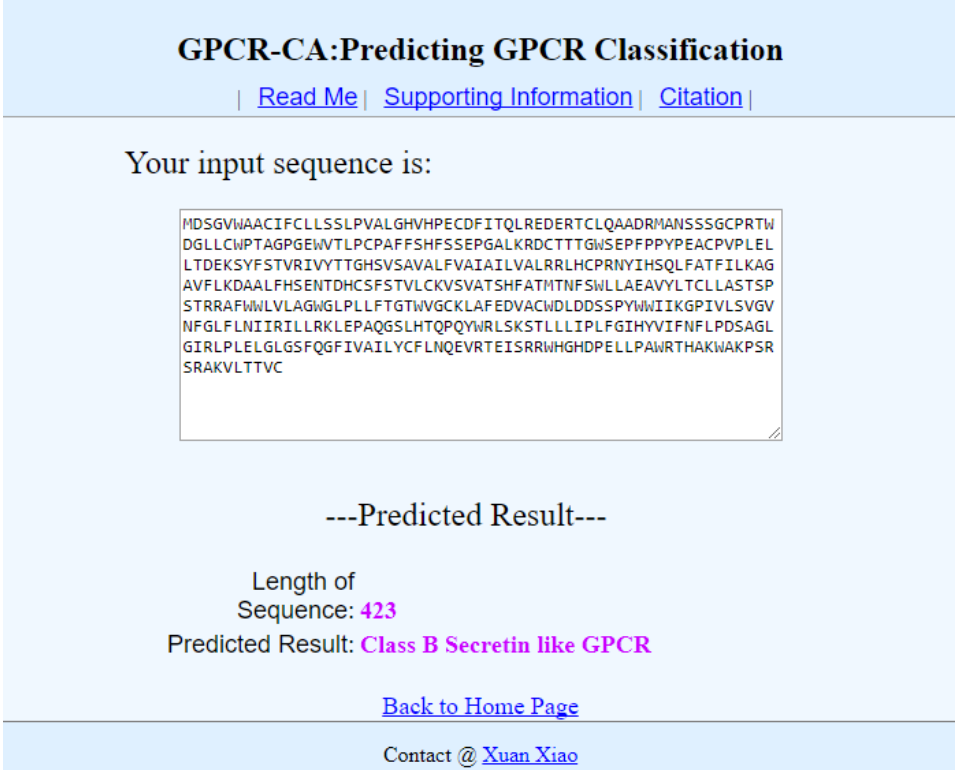


Figure 2.9: GPCR-CA output result for a query sequence

SeQuery

This database [Hu et al., 2019] allows users to visualize the GPCR families' proteome or genome network using a graph-based approach to analyze the relationship of a query sequence with the other GPCRs and their relationship based on their structure, functions from published literature collected from UniProt, RCSB PDB and GPCRdb. The database used 3105 reviewed sequences from UniProt and used 2841 sequences to generate a web-server where all the information of the sequences is used to generate distance matrix using BLAST, then multiple sequence clustering (MSC) to create clusters of the data which is shown in a graphical view using graph-theoretic methods and Cytoscape.js. Currently the database can only take one sequence as an input and shows the closely related proteins and relationship with other GPCRs in various levels. For a given Class B, Secretin like Latrophilin receptor, the web-server provided the following output.

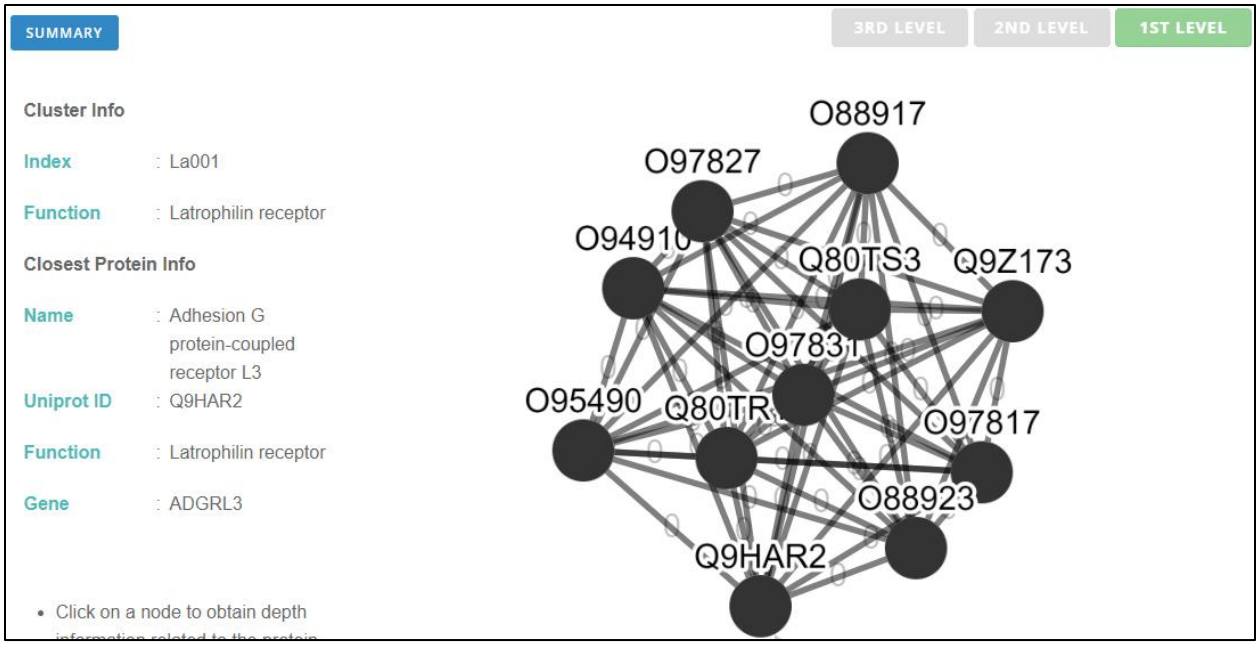


Figure 2.10: Level 1 graphical view of SeQuery

2.3 ALGORITHMS

A protein sequence generally consists of a string of letters in similar or different arrangements, where the letters represent 20 naturally occurring amino acids along with some ambiguous amino acids as well. The prediction of GPCRs from a set of proteins can be approached in several ways that includes using tools to analyze the number of transmembrane helices, observing the alignment sequences and finding motifs or using different machine learning and statistical methods where different protein features are used as the explanatory variables. These tools can be further used to classify GPCRs into lower levels (family, sub-family, sub-sub-family & sub-type). Over the years, diverse set of feature variables and tools have been developed and used by the researchers to not only find novel GPCRs but also understanding the functionality and their signaling pathways. This section gives an overview of how several tools and algorithms have been used and how the important biomedical questions have been addressed.

Search Based tools

The most popular alignment search based tools for distinguishing GPCRs from a given set of proteins are FASTA [Pearson & Lipman, 1988] and BLAST [Altschul et al., 1990]. Both programs consist of a large database and designed to look for sequence similarity, where higher similarity of sequences represent homology of common ancestry. The key difference between these two are that FASTA calculates maximal segment pair (MSP) score by finding similarities in less similar sequences, whereas BLAST measures ungapped alignments and high-scoring segment pair (HSP). While FASTA uses local sequence similarity for the identification of periodic structures, BLAST uses local alignment without gaps.

Other than the search-based methods there are statistical, machine learning and different methods used to predict and classify GPCRs into different families in which the protein sequence similarity is undetermined. Covariant discriminant algorithm [Elrod & Chou, 2002], support vector machines [Karchin et al., 2002], the hidden Markov model [Qian et al., 2003], binary topology pattern [Inoue et al., 2004], protein power spectrum using fast Fourier transform [Guo et al., 2006] and statistical encoding method [Iqbal et al., 2016], bagging classification tree [Huang et al., 2004], *k*-nearest neighbor (KNN) [Gao & Wang, 2006], and the principal component analysis[Peng et al., 2010] are the efficient methods to address the classification problem for higher levels. Combinations of several methods and tools has also been proposed to increase the accuracy for a larger dataset. Selective top-down classifier is also introduced by combining data mining and proteochemometrics [Davies et al., 2008], support vector machines with maximum relevance minimum redundancy and genetic algorithm [Li et al., 2010]. However, in recent studies, different approaches have been developed where family specific motifs [Cobanoglu et al., 2011] or structural region lengths [Sahin et al., 2014] are used for prediction and classification of GPCRs. Other than human genomes, there has been research done for different organisms to reduce the risk of many diseases by using different tools such as TMHMM and GPCRpred for the transcriptome assembly of the cattle tick synganglion [Guerrero et al., 2016]. Furthermore

encoding sequences from the transcriptome of the foreleg [Munoz et al., 2017] has been done using BLASTp, Pfam, GPCRpred, TMHMM, and PCA-GPCR for predicting GPCRs. With every algorithm, there are different sets of features which are studied to increase the efficiency of a certain method, such as amino acid composition, dipeptide composition, various physiochemical properties, cross validation and so on.

Support vector machine

Support vector machines (SVM) are known as the class of statistical learning algorithms that are mostly used for classification problems [Li et al., 2016]. Each data item is plotted in n-dimensional space where n is the number of features with the value of each feature being the value of a particular coordinate. Then the classification is performed by determining the hyper-plane that differentiates between the classes. SVM is very efficient but computationally expensive for classifying GPCRs into higher levels.

Covariant-discriminant algorithm

The covariant-discriminant algorithm introduced by Chou and Elrod (1999) is used to make an analysis of the correlation between the types of G-protein coupled receptors and their amino acid composition. According to the GPCRDB [Horn et al., 1998] the rhodopsin-like amine GPCRs can be classified into six receptors namely acetylcholine, adrenoceptor, dopamine, histamine, serotonin and octopamine. For the training dataset histamine and octopamine receptors were removed as they are very low in quantity. Therefore, the study has been done for the rest of the four types of receptors which gives a 100% success rate on the re-substitution test. This method is efficient if a good training dataset can be established. The overall accuracy using the Jackknife rate for the dataset of 167 GPCRs is 83.23%.

Binary topology pattern

As GPCRs have highly divergent families, Binary Topology Pattern (BTP) [Inoue et al., 2004] is an efficient method for classification and identification with a good accuracy. Inoue et al., described it as a stepwise method where the first step works with three unified functional groups, (i) Class A and Non-GPCR, (ii) Class B + Class C and (iii) Frizzled/Smoothened, with a certain threshold value assigned. In the next step, it works with classifying Class B and Class C, and in step three, Class C is divided into three functional groups followed by Step four, five and six determining the rest of the functional groups along with the identification of the mammalian-type GPCRs. The accuracy is 100% for three groups, and four groups with more than 80%.

Bagging classification tree

Using tree-based algorithm for prediction and classification is very effective and is used to address several classification problems. Huang et al. (2004) describes a bagging classification tree for classifying GPCRs into sub-families and sub-sub-families based on the amino-acid composition. Here it uses the C4.5 algorithm which is used to generate a decision tree. In total 4395 sequences were classified into sub-families and 4036 sequences were classified into sub-sub-families with an accuracy of 91.1% and 82.4% respectively.

K-nearest neighbor

Gao (2006) has introduced a nearest neighbor method to classify GPCRs from non-GPCRs and they further classified into four levels. The classification was done based on the amino acid composition and dipeptide composition of proteins. The dataset consists of 1406 GPCRs for classifying into six families and 1406 globular proteins where the accuracy is measured using the jackknife test and Matthew's correlation coefficient value. The accuracy is improved by using

dipeptide composition for predicting GPCRs from the globular protein than using only amino acid composition. For further classification into six families, both amino acid and dipeptide composition have been used. Comparison of accuracy has been made with other existing methods, and it is observed that the accuracy of the existing method is better than the SVM-based method [Karchin et al., 2002] and covariant discriminant algorithm [Elrod & Chou, 2002]. There are no parameters to be determined for the nearest neighbor algorithm where amino acid and dipeptide composition have been used, this improved the simplicity in classifying GPCRs into four levels.

Principal component analysis (PCA)

Peng et al. (2010) proposed a method called PCA-GPCR for predicting and classifying GPCRs into family, sub-family, sub-sub-family, subtype from a large dataset. The study was done for 1497 sequence derived features which was further reduced into 32-dimensional feature vectors. In first level, the algorithm identifies whether the sequence protein is a GPCR or non-GPCR, then for the GPCRs it is classified into four levels by using the intimate sorting algorithm. The accuracy for the prediction and classification is overall very high.

Family specific motifs

A method has been proposed by Cobanglu et al. (2011) for Class A sub-family classification of GPCRs that uses sequence derived motifs based on the receptor-ligand interaction sites. Distinguishing Power Evaluation (DPE) technique was proposed by Cobanglu et al. (2011) for the classification and also had a discovery of key ligand interaction sites.

Structural regional lengths

Sahin et al. (2014) introduced a method GPCRsort where the length of the transmembrane helices and loop regions was studied for predicting GPCRs using seven feature vectors which

resulted in a fast prediction with a high accuracy of 97.3%. The study has been done on 38,525 protein sequences from GPCRDB and TMHMM is used for identifying the lengths of different regions of the GPCRs.

Statistical encoding method

Iqbal et al. (2016) worked in a statistical distance-based encoding method which works with the various distances of an amino acid in a sequence at different levels of decomposition to form a numeric feature vector. They worked on two different datasets to classify three families and sub-families of Class A. The overall accuracy is more than 94%.

2.4 COMBINATION OF ALGORITHMS

Studies have predicted and classified the GPCRs, where different types of tools are combined together to increase the efficiency for a large dataset or to lower the computational cost. Mostly, it is observed that support vector machines with other tools are combined to get an effective output. This section describes briefly about the tool that has been worked on.

Support vector machine (SVM) with different approaches

Karchin et al. (2002) has used a simple nearest neighbor approach (BLAST), Hidden Markov Model (HMM) and support vector machines (SVMs) which is a group of statistical algorithms for recognizing superfamilies and the small subfamilies of GPCRs that bind the same ligand. The work is focused on comparing different methods with SVMs to observe which one is better computationally. For the classification of GPCRs, the primary sequence information is used which required the extension of the two-class problem to a k-class problem. Karchin et al. (2002) have used the simplest approach to multi-class SVMs by training k one-to-rest classifiers. SVM is computationally expensive but it has significantly less Minimum Error Point (MEP) than the other

methods especially in the case for classifying subfamilies. It is also observed that the higher classification with good approximation can be achieved using SVMtree method. The future work is focused on classifying the subfamilies based on the suitable feature selection for the subfamilies along with the biological knowledge of each subfamily's transmembrane topology.

Yabuki et al. (2005) has described a system called GRIFFIN (G-Protein and Receptor Interaction Feature Finding Instrument) which uses Support Vector Machines and Hidden Markov Model to predict GPCR and G-Protein coupling selectivity along with a hierarchical SVM classifier including the feature vectors to predict Class A GPCRs. For the other type of families (Opsins, Ofactory subfamilies of Class A, Class B, Class C, frizzled and smoothened) HMM is used. As BLAST and FASTA uses sequence similarity for predicting the protein, yet it is not clear to predict GPCRs based on sequence similarity relationship. This system is unique as it uses information from GPCR ligand information and GPCR sequence. SwissProt and TrEMBL databases are used as both ligand and sequence information are present. In total, they have used twenty-four features for ligands and GPCRs.

Another algorithm has been developed by [Liao, Ju, & Zou, 2016] which uses the features from SVM-Prot [Y. H. Li et al., 2016] and Random forest algorithm to identify GPCRs from non-GPCRs with an accuracy of 91.61%.

Fast Fourier transform with support vector machine

Guo et al. (2006) introduced a fast Fourier transform based support vector machine method to classify GPCRs and NRs based on the hydrophobicity of proteins. The three principal properties of hydrophobicity represented by hydrophobicity model, electron-ion interaction potential model and c-p-v model are used to transform the protein sequences into numerical sequences. Three hydrophobicity scales were selected for the optimization as hydrophobicity can vary due to

different experimental conditions, different organic solvents and computing approaches. The dataset used for GPCRs is collected from GPCRDB containing 964 sequences for the final training dataset and for NRs, final training dataset of 465 sequences was observed which is collected from the NucleaRDB. For performance measurement Jackknife test is used as well as for prediction quality, accuracy, total accuracy and Matthew's correlation coefficient are evaluated. Higher accuracy is achieved with the hydrophobicity scale than c-p-v or electron-ion interaction potential model.

Feature selection with support vector machine

A three-layer classifier is proposed by Li et al. (2010) for GPCRs based on the combination of SVM with feature selection method. The method holds high accuracy for classifying into superfamily, family and subfamily of GPCRs. For accuracy measurement Jackknife cross-validation test is used on two non-redundant datasets. Li et al. with 600 hundred features and then used the maximum relevance minimum redundancy to pre-evaluate features and used genetic algorithm is observed to find the optimized feature subset. For developing classification model support vector machine is coupled with the feature selection method. Higher accuracy is observed with the proposed method named GPCR-SVMFS than GPCR-CA and GPCRPred.

Genetic ensemble

Naveed and Khan (2012) introduces GPCR-MPredictor which predicts and classifies GPCRs into five levels (family, sub-family, sub-sub-family, subtype) including the prediction. It is an ensembled approach where *k*-nearest neighbor, support vector machine, probabilistic neural networs, J48, Adaboost and Naives Bayes classifiers have been used. Amino acid composition, pseudo amino acid composition and dipeptide composition these three features are used to predict

and classify GPCRs. This ensembled approach has a higher accuracy than principal component analysis (PCA) method in all the five levels.

2.5 ACCURACY AND SUMMARY

With different sets of algorithms and dataset, set of measurements have been taken to identify the accuracy for predicting and classifying GPCRs. Cross-validation and Jackknife test are the most commonly practiced methods along with sensitivity and specificity to measure the performance of a predictive model. The following table shows the information of the accuracy observed from different methods for a certain dataset.

Table 2.1: Summary of datasets, algorithms and accuracy measurements

Name	Dataset (sequences)	Prediction/Classification	Accuracy measurement	Accuracy
Covariant-discriminant [(Elrod & Chou, 2002)]	GPCRDB: 167	Correlation between GPCRs and amino acid composition	Re-substitution test Jackknife	Success rate 100%
Binary topology pattern [(Inoue et al., 2004)]	SwissPROT: 954 Mammalian GPCRs: 811 Non-GPCRs: 17	Ten groups	Sensitivity Specificity Success rate	100% for three groups >80% for four groups 65.8% for Non-GPCRs
Bagging classification tree [(Huang et al., 2004)]	GPCRDB: 8431	Sub-families Sub-sub-families	Cross validation	91.1% for sub-family 82.4% for sub-sub-family
Fast Fourier transform [(Guo et al., 2006)]	GPCRDB: 946 (GPCRs)	(Class A to Class E)	Accuracy Total accuracy Matthew's correlation coefficient (MCC)	>94% accuracy except for Class C. >92% MCC for all the classes
GPCRTree [(Davies et al., 2008)]	GPRDB: 8222	Class Sub-family Sub-sub-family	GPCR servers	97% for class 84% for sub-family 75% for sub-sub-family

SVM & feature selection [(Z. Li et al., 2010)]	GPCR-CA: 730	Prediciton Class (A-E)	Jackknife Cross validation	98.22% for prediction 96.99% for class
SVM-Prot features & Random forest	Uniprot: 5026 GPCRs, 10386 non-GPCRs	Prediction	Cross validation Sensitivity Specificity Accuracy Matthew's correlation	91.61%
Principal component analysis (PCA) [(Peng et al., 2010)]	GPCRDB: 14026	Prediction Family Sub-family Sub-sub-family Subtype	Jackknife	99.5% for prediction 88.8% for family 80.47% for sub-family 80.3% for sub-sub-family 92.34% for subtype
Family specific motifs [(Cobanoglu et al., 2011)]	GPCRDB: 4889	Class A subfamilies (Amine, Peptide, Rhodopsin, Prostanoid, Olfactory)	Cross validation	90.7% overall
GPCR-MPredictor [(Naveed & Khan, 2012)]	GPCRDB and PCA: 14026	Prediction Family Sub-family Sub-sub-family Subtype	Jackknife test Sensitivity Specificity Matthew's correlation F-measures	99.75% for prediction 92.45% for family 87.80% for sub-family 83.57% for sub-sub-family 96.17% for subtype
Structural region lengths [(Sahin et al., 2014)]	GPCRDB: 38525	Prediction And GPCRDB classifications	Cross validation	97.3%
Statistical encoding method [(Iqbal et al., 2016)]	GPCRDB: 2925	Class (A-C) Class A families: Dopamine, Serotonin, Chemokine	Cross validation	94-98%

Chapter 3: Compiled Datasets

3.1 GPCR & NON-GPCR DATASETS

The databases that are more likely being used to collect GPCR sequences are GPCRdb and UniProt-KB/Swiss-Prot. We have initially used Swiss-Prot database as it clearly identifies the selection of confirmed (reviewed) or predicted (unreviewed) protein sequences. If a particular dataset contains predicted (unreviewed) proteins, some of the data might become obsolete over the time period or be confirmed as another type of protein as well. Therefore, the collection of confirmed GPCRs is done using the Swiss-Prot database.

There are 3129 full length and fragment sequences collected from all the families of GPCRs. Following shows the number of sequences available in different families:

Table 3.1: The number of GPCR sequences collected for each family

Family name	Number of sequences		Number of organisms
	Full Length	Fragments	
Rhodopsin (Class A)	2358	135	308
Secretin (Class B1)	113	0	37
Adhesion (Class B2)	91	1	11
Glutamate (Class C)	112	0	23
Fungal (Class D)	13	0	6
cAMP (Class E)	11	0	1
Frizzled (Class F)	79	6	12
Taste2 (Class T2R)	211	0	12

There are two types of non-GPCR dataset available in our database, non-transmembrane non-GPCRs and the transmembrane proteins that are not GPCRs as well. It is very important to have a diverse set of proteins as a negative test set to assess any prediction algorithms, currently the dataset consists of 1331 non-transmembrane and 2244 transmembrane non-GPCRs.

To understand the reasoning behind ligand binding and the function of different signaling pathways of GPCRs, other model organisms (Langenhan et al., 2015) are often used. As most of algorithms and tools are focusing on predicting or classifying GPCRs in human, here the predicted GPCRs from *Drosophila melanogaster* (Fruitfly), *Anopheles gambiae* (Mosquito), *Rhipicephalus microplus* (cattle tick) and their closely related organisms are incorporated in the database that are collected from Swiss-Prot and other sources. The dataset is refined by only keeping the sequences consisting of 6 to 8 helices and all the features are incorporated in the database.

UCLUST

This algorithm was introduced by Edgar (2010) which is clustering sequences based on the similarities with a given identity threshold, T. UCLUST provides constructive results when protein sequences are given T>~50% or for nucleotides T>~75%. Therefore, to check the homology in our GPCR dataset, different threshold values from 50-90 have been used to identify the diversity on all the full-length confirmed GPCR sequences which will be very useful for the users for training and testing their algorithm with our given dataset. The following table shows the number of clusters constructed with varied threshold values. It is observed that the number of clusters are comparatively higher even with the less threshold value which shows the diversity of the dataset.

Table 3.2: Number of clusters using UCLUST with different identity threshold values

T=50% Uniques: 2944 Singletons: 2912 Clusters: 677	T=60% Uniques: 2944 Singletons: 2912 Clusters: 910	T=70% Uniques: 2944 Singletons: 2912 Clusters: 1117
T=80% Uniques: 2944 Singletons: 2912 Clusters: 1418	T=90% Uniques: 2944 Singletons: 2912 Clusters: 1927	

3.2 CLASSIFICATION SYSTEM OF GPCRDB

This database is well known as a data source dedicated to collecting, analyzing, and validating data on GPCRs as well as other useful structural and signaling pathway information. The database dedicates a large part to provide information on experimental data to help researchers explore and test their work. Since, we only collected the three lower levels (subfamily, sub-subfamily and subtype) classification information from the database, the background on the collection of the sequences is highlighted here.

Initially, GPCRdb [Horn et al. 1998], started with more than 800 sequences from the SWISS-PROT database [Bairoch and Apweiler, 1997] to retrieve the best curated collection of protein sequences. Then in three years [Horn et al., 2001] database collected more than 2000 sequences including fragments. By 2003, GPCRdb [Horn et al., 2003] had tripled the number of sequences, collected from Swiss-Prot and its computer annotated supplement, TrEMBL. During these three releases of the GPCRdb, the GPCRs were divided into five major classes with Class A,B,C, Frizzled/Smoothened and Non-GPCRs [Inoue et al., 2004] with a varying number of minor sub-families to be 151, then 222 and 283 respectively. Due to the absence of large number of sequences that we have today, the functional sub-families of the classes were differently categorized which are now mostly included in the sub-sub-family levels of the database.

To incorporate more information (e.g. structural, mutation, ligand-binding etc.) and enhance the power of computational tools, GPCRdb [Vroling et al., 2011] incorporated more than 27,000 sequences which was collected from NCBI's non-redundant (NR) protein sequence database [Pruitt et al., 2007]. Initially the number of sequences were more than 80,000 which they reduced by using a database of Hidden Markov Model. With this large number of data, the total number of families including multiple sequence alignment (MSA) became 1270. Up to this stage when web browsers were not introduced, users could only get data or alignments via an automated emailing system.

By 2015, family level classification also included Vomeronasal (VR1) and Taste 2 receptors as separate classes along with previously described Class A, B and C [Hu et al., 2017]. The Taste 2 receptors were included as a separate class/family as they have more similarity with Class A receptors [Nordström et al,, 2011] that was previously recognized as a sub-family of Frizzled. Around 2016, GPCRdb came up with an immense amount of changes with their web services, analyzing tools and classification system with lesser experimental data than the last release. The Vomeronasal as a family, and the olfactory receptors as a sub-family of Class A were completely excluded, whereas they started more focusing on the ~350 non-olfactory receptors found in human GPCRs [Munk et al., 2016] that have a major impact on the prescription drugs. At this point, data is also easily downloadable using the Django web framework [Pándy-Szekeres et al., 2018] and available for the 7 major families and 500 lower levels of classification.

The GPCR classification levels can be arranged into five levels among which the first two levels represent the prediction of a protein as a GPCR and then categorizes them by using the family level classification systems GRAFS, where Fungal pheromone (Class D) and cAMP (Class E) receptors are not included. The family level classification system is further incorporating two more classes, Taste 2 and Orphan receptors. Then the lower level of classifications such as level 3, 4 and 5 also known as sub-family, sub-sub-family and sub-type respectively. For these levels, GPCRdb categorizes the proteins based on their ligand and receptor binding properties rather than sequence homology, as a result few sub-families may have sequence similarity but are not categorized in the same sub-family due to functional dissimilarity and vice versa [Karchin et. al, 2002]. Currently, there are more than 15,000 GPCR sequences in GPCRdb which comprises of both reviewed and unreviewed (predicted data) sequences.

GPCRdb maintains a four-segment number system to define all the proteins based on every unique family, sub-family, sub-subfamily and sub-types where they also include the UniProtKB IDs as well. The sub-family is usually based on the endogenous ligand type, sub-sub-family represents the receptor family followed by the sub-type. For example, if a protein has an GPCRdb ID 001_001_003_009, it means starting from the left, the 001 indicates the family level which is

Rhodopsin, 001_001 is the sub-family named Aminergic receptors, 001_001_003 is the sub-sub-family as Adrenoreceptors and the whole four-segment of 001_001_003_006 represents the sub-type of α_{2C}-adrenoreceptor. If the number is 001_001_003_009 then it represents everything as same as the above example but the sub-type is β_3-adrenoreceptor. Following shows the number of classification levels available in GPCRdb.

Table 3.3: Number of classification levels in GPCRdb

Level 2 (Family)	**Level 3 (Sub-family)**	**Level 4 (Sub-sub-family)**	**Level 5 (Sub-type)**
001 – Rhodopsin	11	61	287
002 – Secretin	1	5	15
003 – Adhesion	1	9	33
004 – Glutamate	3	5	21
005 – Frizzled	1	1	11
006 – Taste 2	1	1	25
007 – Other GPCRs	1	1	6

It is observed that from the 3129 GPCRs that is available in our database, 2011 sequences matched with GPCRdb, and as all the sequences have the lower level classifications, our database accommodates classification information on those 2011 sequences. The next section discusses about the sequences that matched and the ones that didn't.

3.3 COMPARISON OF GPCRDB AND GPCR-PENDB

GPCRdb incorporates a large number of predicted data and their corresponding information that focuses on the structure, G proteins, ligands and mutants etc. All the sequences contain the information on their genes, functions and receptors for the data as well. Although our database, GPCR_PEnDB contains much less sequences than GPCRdb, the reason that only ~64% (Table 3.5) data matched with the database is because GPCRdb does not incorporate any reviewed

or unreviewed GPCR sequences if they are missing any information about gene, function, receptor or ligand, which would also result in a missing information to create a four segment number code it uses. Following table shows the number of sequences present for each Class/family in the database and how many of them matches with GPCRdb.

Table 3.4: The number of sequences that are found in GPCRdb and GPCR-PEnDB

Class ID	Class	Number of sequences		Number of Matches
		GPCRdb	GPCR-PEnDB (Reviewed GPCRs from Swiss-Prot)	
001	Class A (Rhodopsin)	11732	2493	1551
002	Class B1 (Secretin)	850	113	69
003	Class B2 (Adhesion)	544	91	81
004	Class C (Glutamate)	969	112	83
005	Class F (Frizzled)	47	85	46
006	Taste2R	610	211	175
007	Other GPCRs	9	4	4

The Class A (Rhodopsin-like) is the largest family of GPCRs, where some of the major sub-families or sub-sub-families like olfactory/odorant, some of the opsins, vomeronasal, tyramine, several octopamine and viral G protein-coupled receptors are not present in GPCRdb. All the GPCRs have the location of the seven transmembrane helices according to

UniprotKB/Swiss-Prot, some of the receptors like Ryamide, RPE-Retinal, QRFP-like peptide, Pyrokinin-1 receptor are also predicted by TMHMM 2.0 as well with seven transmembrane helices. We have at least one sequence from all the sub-types from Class A that are available in GPCRdb and a list of names for all the sub-families that are present in either GPCRdb or GPCR-PEnDB or both are shown. In Class B1 (Secretin) some of the major sub-families like Calcitonin, Glucagon, different hormone receptors are present in both, whereas some of the Calcitonin receptors are not present in GPCRdb along with Methuselah and different Latrophilin receptors. For Class B2, several types of Adhesion G protein coupled-receptors are present, whereas some of the receptors are only available in GPCRdb for human or other species but not for rat, mouse or dog. As GPCRdb contains the unreviewed proteins, some of them became obsolete according to Swiss-Prot as well. For Class C, Taste 1 receptor is present in both databases, whereas some of the groups of Metabotropic glutamate receptors are present in both but some are missing in GPCRdb. For Class F (Frizzled) and Class T2R, GPCRdb has less sequences than our database, as the function (ligand, receptor binding etc.) of those proteins yet to be confirmed. There are also some groups where GPCR-PEnDB has sequences from both rat and mouse but GPCRdb has from only one organism. The complete list of receptors that are present in both databases and missing in one another is shown in the Appendix M.

It is observed that GPCRdb has only 9 sequences in the family 007 (Other GPCRs/Orphan GPCRs), out of which one of them belongs to Class A (Rhodopsin), also known as a putative olfactory receptor and three of them belongs to Class B1 (Secretin) also known as GPR157. These proteins have sequence similarities with multiple classes and that is why they are categorized as Orphan receptors. Therefore, a separate class is created for these types of proteins which shares similarities with more than one family.

3.4 DATA COLLECTION METHODS

This section describes the methods that are used to collect the protein sequences for both GPCRs and Non-GPCRs. Then it also discusses how the lower level classification is obtained from the GPCRdb.

3.4.1 Collection of the GPCRs and Non-GPCRs

In the UniProtKB/Swiss-Prot database, instead of searching for "G protein-coupled receptors", the advanced search option was used where user can search for a specific type of protein family. At first we followed the IUPHAR classification system to find the six families/classes of GPCRs and then extended the search to look for another family of GPCRs that is not used in the classification system called Taste2 receptors. This family has the second largest number of confirmed GPCRs after the Rhodopsin family.

The first five protein families, Rhodopsin, Secretin/Adhesion, Glutamate, Fungal and cAMP receptors can be searched in the Swiss-Prot database using the assigned family numbers 1,2,3,4 and 5 respectively. To get the sixth family, Frizzled, the search used was "G-protein coupled receptor fz smo family" and all the family datasets were downloaded separately and then merged. Then by using "Taste2 receptors and G protein coupled receptors", the protein sequences are collected and carefully analyzed to only get the confirmed Taste2 receptors.

For the non-GPCRs, all the protein sequences are collected from Swiss-Prot and run through CDHIT with the threshold set at 0.50 which resulted in a more diverse set of proteins as CDHIT clears out the sequences with greater than or equal to 50% sequence identity. After collecting and parsing the data (Appendix B) all the non-transmembrane non-GPCRs sequences, it is checked using the property "transmembrane" in Swiss-Prot where it shows if a protein has any transmembrane helices or not. For the transmembrane specific non-GPCRs, it is observed the number of transmembrane helices vary from 1 to 24, including a total of 137 proteins that have 7 transmembrane helices but do not have the same functions as GPCRs.

3.4.2. Collection of families from GPCRdb

To get the lower level classification information on all the proteins that are available in GPCRdb and then sorting these with our existing confirmed GPCR dataset, several steps were followed. At first, collecting the names of all the families available, and then find their corresponding list of sub-families, sub-sub-families and sub-types. Following shows how the names of the classes and sub-families are displayed in GPCRdb, that can be downloaded and then parsed.


```
{
    "slug": "001_001",
    "name": "Aminergic receptors",
    "parent": {
        "slug": "001",
        "name": "Class A (Rhodopsin)"
    }
},
{
    "slug": "001_001_001",
    "name": "5-Hydroxytryptamine receptors",
    "parent": {
        "slug": "001_001",
        "name": "Aminergic receptors"
    }
},
{
    "slug": "001_001_001_001",
    "name": "5-HT<sub>1A</sub> receptor",
    "parent": {
        "slug": "001_001_001",
        "name": "5-Hydroxytryptamine receptors"
    }
},
{
    "slug": "001_001_001_002",
    "name": "5-HT<sub>1B</sub> receptor",
    "parent": {
        "slug": "001_001_001",
        "name": "5-Hydroxytryptamine receptors"
    }
},
```


Figure 3.1: GPCRdb family IDs and names

After parsing and organizing the four-segment numbers and their corresponding names the following number of classification levels are found for each family (Appendix A).

Then using the IDs for the family level (e.g. 001) the files were downloaded and parsed to get the UniProt IDs and the corresponding four segments numbers from the following format.

Figure 3.2: GPCRdb format for available sequences

In this way all the sequences from GPCRdb are collected and then it was compared using the Uniprot IDs with our dataset to check which of the GPCR sequences match and then organized like the following table for the all the classes individually to analyze and then later upload it in the database.

Table 3.5: Example of the file created with the lower level classification for each class

ID	Sub-type ID	Sub-type name	Sub-sub-family ID	Sub-Sub-family name	Sub-family	Sub-family name	Family
P35404	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)
P60020	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)
P28564	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)
P79250	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)
O08892	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)

P49144	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)
P46636	001_001_001_002	5-HT_{1B} receptor	001_001_001	5-Hydroxytryptamine receptors	001_001	Aminergic receptors	Class A (Rhodopsin)

Chapter 4: GPCR-PEnDB

This chapter focuses on the contents of the GPCR-PEnDB and how the connection is established between the relational database and the web-server. GPCR-PEnDB relational database contains the features, family classification, source organisms and the functions of GPCRs and non-GPCRs. These data can be accessed, analyzed and used in various forms. There are several options available in the web-server that allows user to observe and collect data on one particular type of protein to a variety of protein groups. GPCR-PEnDB focuses on providing users a constructive training and testing dataset for studying the efficiency of computational algorithms. This setup also links the proteins to their corresponding UniProtKB protein description pages as well as the 3D structures that are available in the RCSB PDB (Protein Databank). Also, users can download the result in both CSV format and the Fasta sequences using the similarity tool CD- HIT.

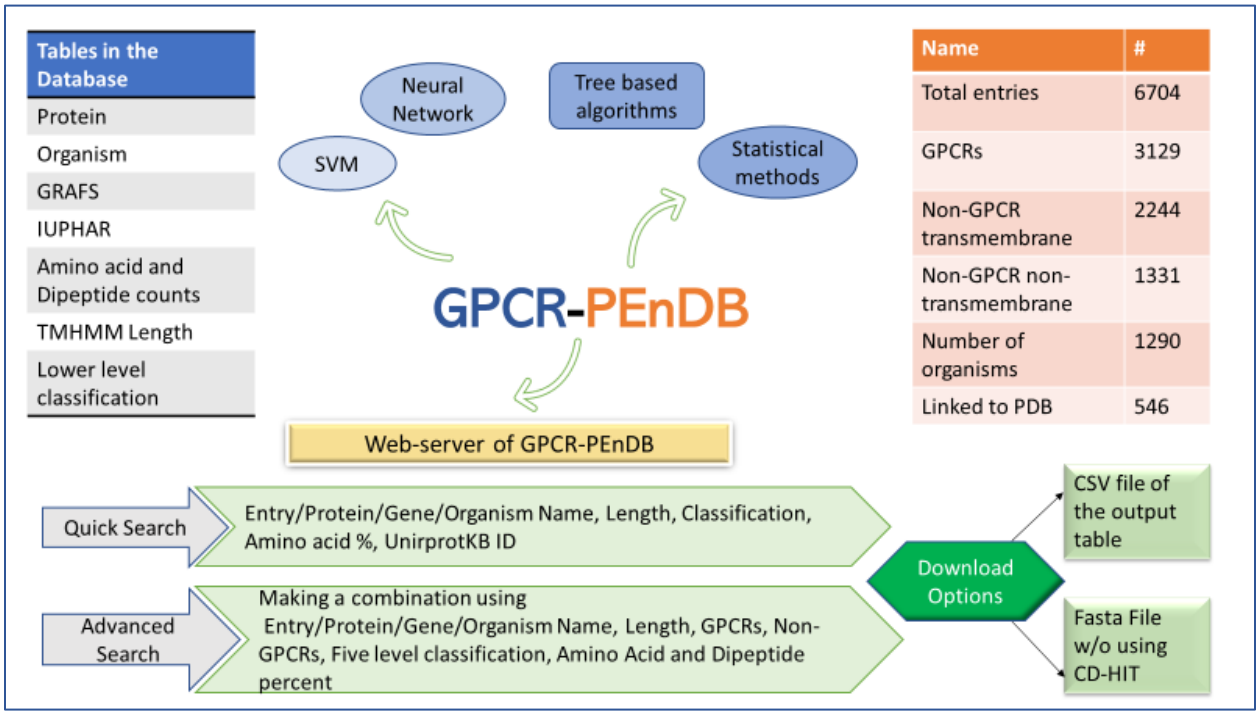


Figure 4.1: The architecture of GPCR-PEnDB

4.1 GPCR-PEnDB Database

GPCR-PEnDB is a relational database where data can be accessed in many ways without reorganizing the tables that contain information about each protein starting from a general overview (e.g. name, id, gene), then different levels of classification, source organisms and protein features. In order to easily access information on both the positive and negative datasets we have created seven tables, namely Protein, Organism, AA_Dipeptide, TMHMM_Length, IUPHAR, GRAFS and LL_classification. The entity relationship diagram is shown in Figure 4.2.

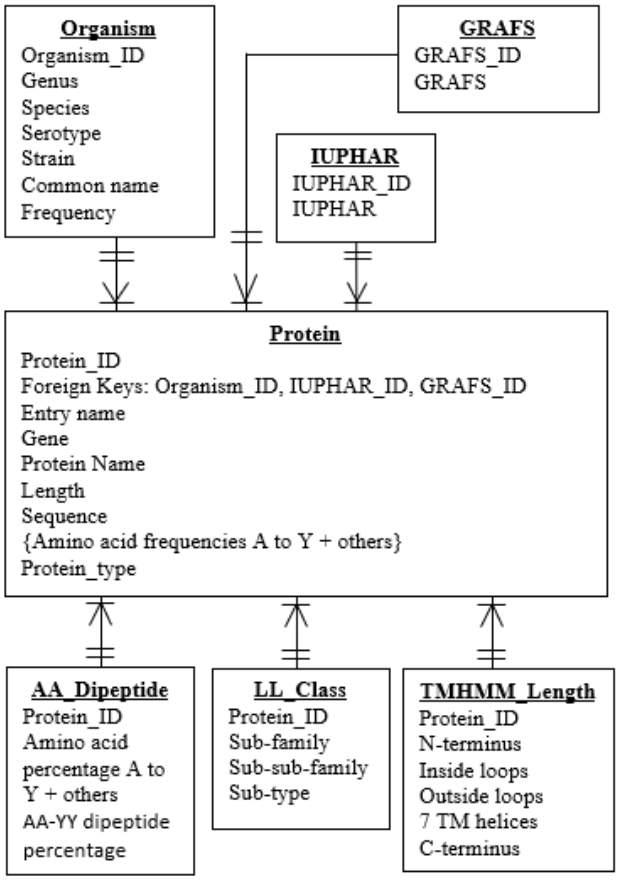


Figure 4.2: Entity Relationship Diagram

The Protein table (Primary key: Protein_ID) is the parent table containing the sequence ids, protein names, entry names, lengths of each sequence (in terms of number of amino acid residues), the amino acid percentages and the indicator distinguishing GPCRs from non-GPCRs. The ID number, entry, protein and gene names are collected or followed as it is in the Swiss-Prot database

by downloading its spreadsheet and after re-arranging it, the field "Protein_type" is added manually where GPCRs, GPCR fragments, transmembrane specific non-GPCRs and non-transmembrane GPCRs are labeled as CG, CGF, CNT and CN respectively, whereas for the predicted sequences based on the type of the protein it can be either PG (Predicted GPCR) or PN (Predicted Non-GPCR). The PDB IDs are also included in a separated field for only the GPCR structures available in RCSB PDB. The amino acid percentages are calculated using a python script (Appendix B) and three other different IDs (foreign keys) have been assigned to the protein sequences based on the GRAFS and IUPHAR systems along with the IDs assigned for the organism types. These allow the proteins to connect with the GRAFS, IUPHAR and Organism tables respectively using the foreign keys defined in the table as GRAFS_ID, IUPHAR_ID, Organism_ID. For simplicity, the Protein_ID is used as the foreign key in the tables containing amino acid, dipeptide percentages and predicted regional length by TMHMM. As the Protein table contains 32 fields, it is very important to specify all the fields while adding new data even if the fields are not applicable. In this case, we have used a hyphen '-' to indicate a non-applicable or an unknown field.

In the Organism table (Primary key: Organism_ID), all the entities have their scientific names and common names along with an identification number. For bacteria and viruses, serotype and strain information are also included. An additional column named "Frequency" has the counts of the sequences available in the dataset for each type of organism. The Organism IDs are assigned using a python code (Appendix C). With this structure, user can construct datasets that focus on a set of specified organisms.

The GRAFS and the IUPHAR tables (Primary keys: GRAFS_ID, IUPHAR_ID) have the same structure with two columns. The first column contains the IDs and the second column has the family names (also known as level 2 classification) of the classification system as shown in Table I. As the further level classification information is not available for all the proteins, it has been kept separated in the LL_Class table. In this table the primary key is the Uniprot IDs of the GPCRs and three fields indicating the protein functions based on their lower level classification.

The fields are sub-family, sub-sub-family and sub-type that are also known as the level 3, level 4 and level 5 classification respectively.

The AA_Dipeptide table (Primary key: Protein_ID) contains amino acid and dipeptide percentages calculated using python scripts (Appendix D and E). It has 423 columns, with the first one containing the protein name. The next 20 columns give the percentages of the common types of amino acids (represented as A, C, D,…,W, Y) and one more column for all other unidentified amino acids found in the sequences. These are followed by the percentages of the 400 dipeptides (AA, AC, AD, …., YW, YY) plus one column for all unidentified dipeptides.

GPCR structural features that include the number of transmembrane helices and their lengths, the lengths of the N- and C-termini, as well as the inside and outside loop lengths are important characteristics for prediction and classification. As from a recent study it is found that only 105 unique GPCR structures are available in PDB [Bernal et al., 2000], we decided to use the hidden Markov model based transmembrane helix prediction tool TMHMM2.0 [Sonnhammer et al., 1998] to estimate of the lengths of the structural regions for the GPCR dataset and generated the TMHMM_Length table (Primary key: Protein_ID). The tool is installed locally and the output is parsed using a python script (Appendix F) where all the lengths of different regions are parsed and kept in separated columns. This table contains the predicted lengths of N- and C-termini, seven transmembrane helices, three inside and three outside loops for the GPCRs whenever the estimation is possible.

The GPCR-PEnDB database resides on apps02.bioinformatics.utep.edu which is a Dell PowerEdge R430 rack server that uses dual Intel Xeon E5-2620 processors (6 cores at 2.4GHz for each processor) and has two 16Gb DIMM memory modules. The server utilizes the CentOS 7 operating system, a Red Hat Enterprise Linux derivative, and is part of the Bioinformatics Network at UTEP. The physical server is located within the Research and Academic Data Center. The database is built in MySQL Version 14.14 Distribution 5.6.37, for Linux(x86_64) using EditLine wrapper.

4.2 GPCR-PENDB WEB-SERVER

Our team has developed the GPCR-Prediction Ensemble (GPCR-PEn) pipeline http://gpcr.utep.edu/ where the algorithms that are available with source codes are incorporated to make an ensemble approach. Currently we have BLAST, Pfam, GPCRPred and GPCRTm incorporated where user can use one or multiple fasta sequences and get a result predicted by the combination of selected algorithms.

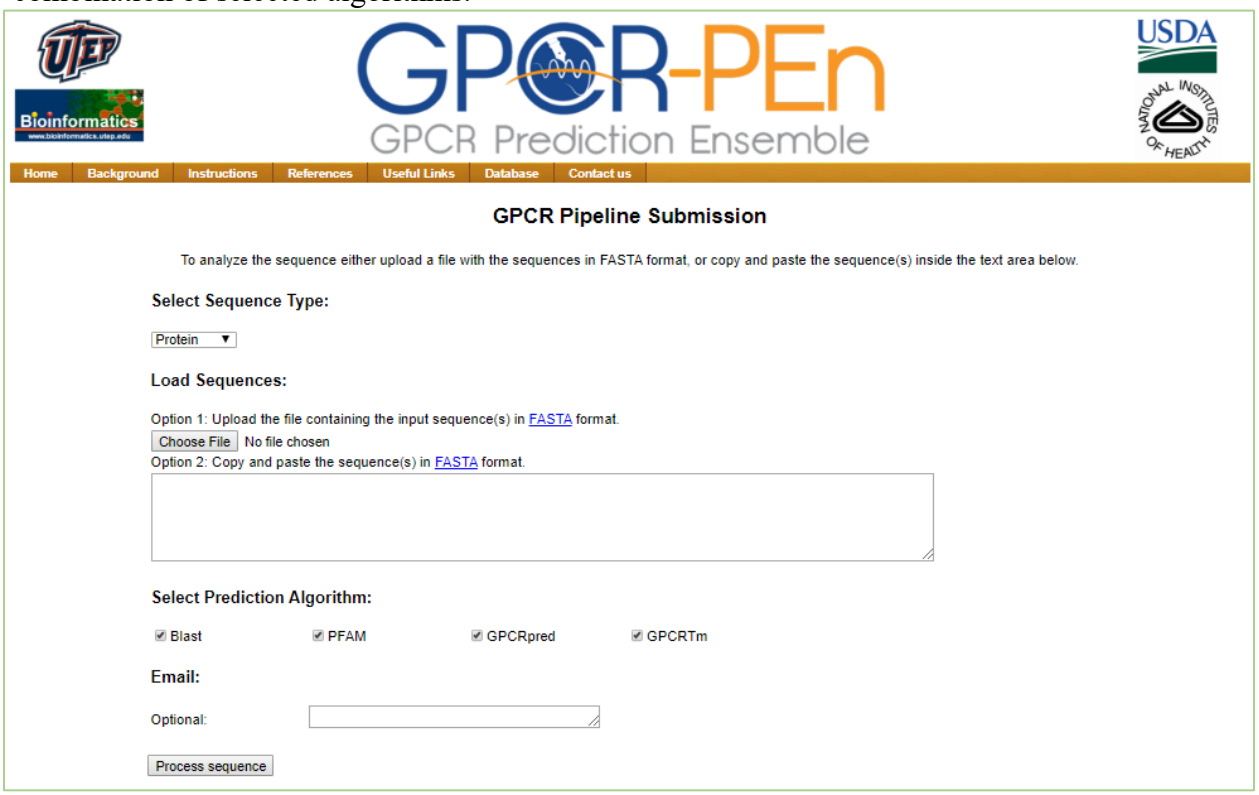


Figure 4.3: GPCR-PEn interface

Within GPCR-PEn, a web interface for the GPCR-PENDB is implemented in the web.py framework 0.37 version, has been made publicly accessible at http://gpcr.utep.edu/database. This web server (Appendix H) allows users to generate MySQL queries conveniently by specifying different input search parameters. Two search options are available. The quick search allows users to specify only one conditional clause (MySQL clause name: WHERE) from only a single table. The output will display information from all the tables for those proteins satisfying the search condition. Following Figure 4.4 shows the options available for the quick search, these options

will pull up information from the database and will display the result in a table. Then the saved inputs are used twice, one for writing the results in a file in CSV format and to assemble the protein sequences in a separate file in fasta format. Links are given to download both type files. For the fasta file, we have also provided the ability to use the CD-HIT tool (Appendix I), so that users can either download all the sequences or cluster similar sequences using a similarity threshold value between 0.65 and 1.0.

For example, if a user wants to explore all the human data available in the database including GPCRs and non-GPCRs, in the first field if the user specifies the name of the organism and select the Organisms field from the drop down menu, it will display all the data available in a table, where user can download the table to get the information about the proteins, their functions, amino acid percentages, TMHMM predicted regional lengths, classification as well as, can only download the FASTA sequences for those particular proteins.

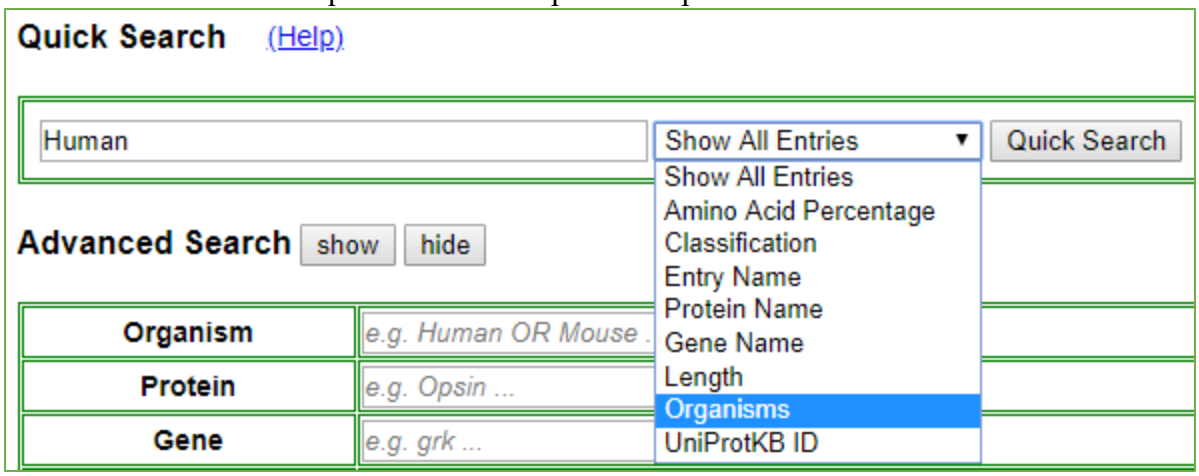


Figure 4.4: Quick search options available in the web-server

In the advanced search, multiple conditional clauses can be specified by the user to generate the query. The search options cover almost all the fields of the database, which makes the web-server efficient to collect any amount of data based on any criteria or selection. The interface is designed in such a way that users who are particularly interested just to look for a set a of proteins from a biological aspect can also make quick or filtered searches as well. Users can either look for a specific group of organisms, sequences that hold a certain amount or type of amino acid residues,

classification-based search or even creating an own combination of choice. As a simple example if users want to download the sequences without the presence of any special characters, they can specify “Others = 0” (count of special character is zero), in the “Amino Acid percentage” field, that will only display the complete sequences without any special characters.

We have used scripts in python (Appendix I) to save and use the inputs specified by the user and convert it into an SQL query (Appendix J) to accumulate the results from the database and show it in the result page (Appendix K) of the webserver in a tabular format.

As mentioned before, if a user uses Quick search, it pulls up all the sequences based on only one condition (e.g. Organism name human) but if users want to create an intricated query such as looking for Aminergic receptors in human and rat where the lengths of the sequences are greater than 500 amino acid residue just to compare these two organisms’ Aminergic receptor properties.

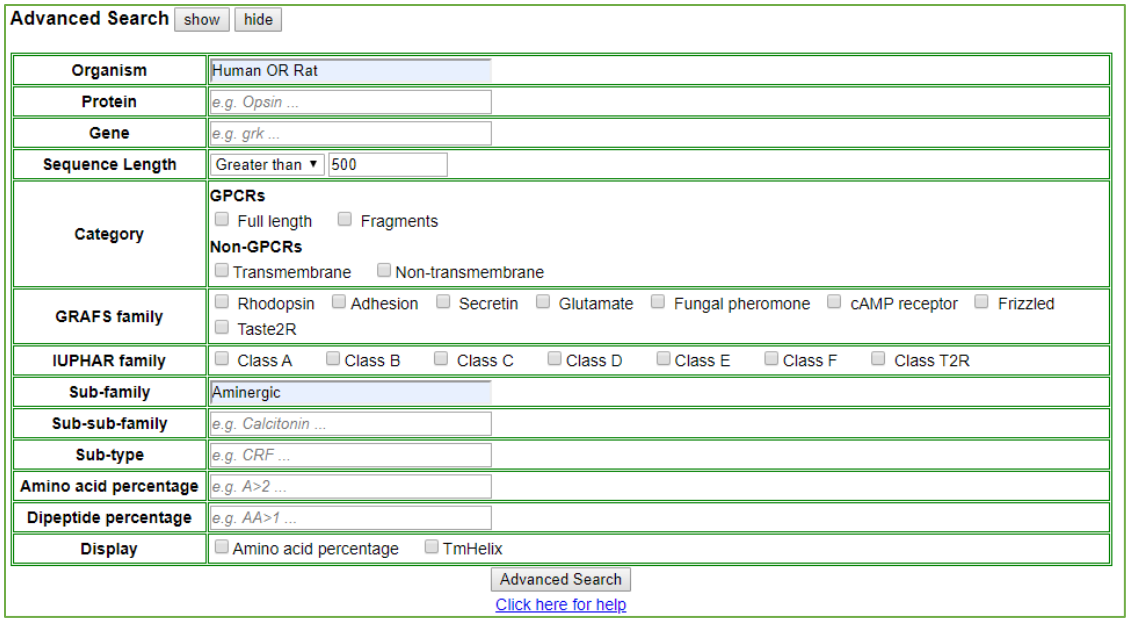


Figure 4.5: Creating an advanced search query

The above set up of advanced searches covers a wide range of queries that can be generated for retrieving data from a database even without having prior knowledge about the programming language or the schema that has been used. This web-server is designed in a such a way where users can specify any set of fields based on their preference stored as inputs and the SQL query

will be generated in the back end that displays the resulting query and provides options to save the data according to the users specifications.

Chapter 5: Searching GPCR-PEnDB

5.1 MySQL query to GPCR-PEnDB

We have gathered the general information (e.g. protein name, gene name, sequence) for each protein as well as the common features like amino acid and dipeptide counts and percentages, the five-level classification for the available GPCRs and the length of different regions using TMHMM 2.0. We encountered some problems with the TMHMM estimation where the predicted number of helices in a GPCR differed from seven. As TMHMM is a prediction tool, some of the confirmed GPCRs got predicted with 6 or 8 transmembrane helices, so we kept them separate, and provided the regional lengths only for the GPCRs that were predicted with 7 transmembrane helices. Ideally, these structural features should be obtained from the actual 3D structures deposited in structural database like PDB (Protein Data Bank) [Berman et al., 2000].

Having the searchable database GPCR-PEnDB enables us to generate MySQL queries consisting of various clauses that not only involve joining multiple tables but grouping the results based on a numeric range. The following figure contains a query that search for all the GPCRs with 'Class A' IUPHAR classification, >10% serine in amino acid composition, and >300 in C-terminal length. The search result, as shown in Table 5.1, also displays the UniProt Protein ID, sequence length, and the source organism.

```
'SELECT Protein.Protein_ID, Protein.S,
Protein.Length, TMHMM_Length.C_term,
Organism_v2.Common_name

FROM Protein

INNER JOIN IUPHAR ON IUPHAR.IUPHAR_ID =
Protein.IUPHAR_ID

INNER JOIN TMHMM_Length ON
TMHMM_Length.Protein_ID = Protein.Protein_ID

INNER JOIN Organism_v2 ON
Organism_v2.Organism_ID=Protein.Organism_ID
```

Figure 5.1: Example of a MySQL query to GPCR-PEnDB

Table 5.1: Output table of the query asking for GPCRs in Class A with >10% serine and C-terminal length > 300.

Protein_ID	Serine(S) %	Length	C_term	Common name
Q9W534	10.91	670	305	Fruit fly
Q6NV75	10.18	609	311	Human
Q86SP6	12.31	731	367	Human
Q8K0Z9	10.30	631	333	Mouse
Q9DDD1	12.31	723	357	Chicken
Q924Y8	11.92	730	368	Rat

One other important advantage of the MySQL database is that it allows us to add new sequence collections (e.g., the collection of predicted but unconfirmed GPCRs as mentioned above) easily. Furthermore, we can regularly update the sequence collection so that users can access the most current information.

5.2 SEARCH VIA GPCR-PENDB WEB-SERVER

The web interface (Figure 5.2) is designed to provide the flexibility for users to obtain information from various entities without constructing MySQL queries. One can assemble different queries and narrow down the search to accumulate details about the entities of interest by entering or selecting parameters on the webpage. Each resulting protein ID is linked to the UniprotKB database for the user to find more detailed information about the protein. The user can also specify whether the display should include detailed information of amino acid percentages or the structural region lengths estimated by TMHMM. From resulting page, users can compile and download the results in both CSV and FASTA formats. The FASTA formatted download can be done with or without using the clustering tool CD-HIT that provides non-redundant representative sequences as output. This allows user to keep the fasta sequences separated from the sequence derived features so that other features can be generated and used if needed.

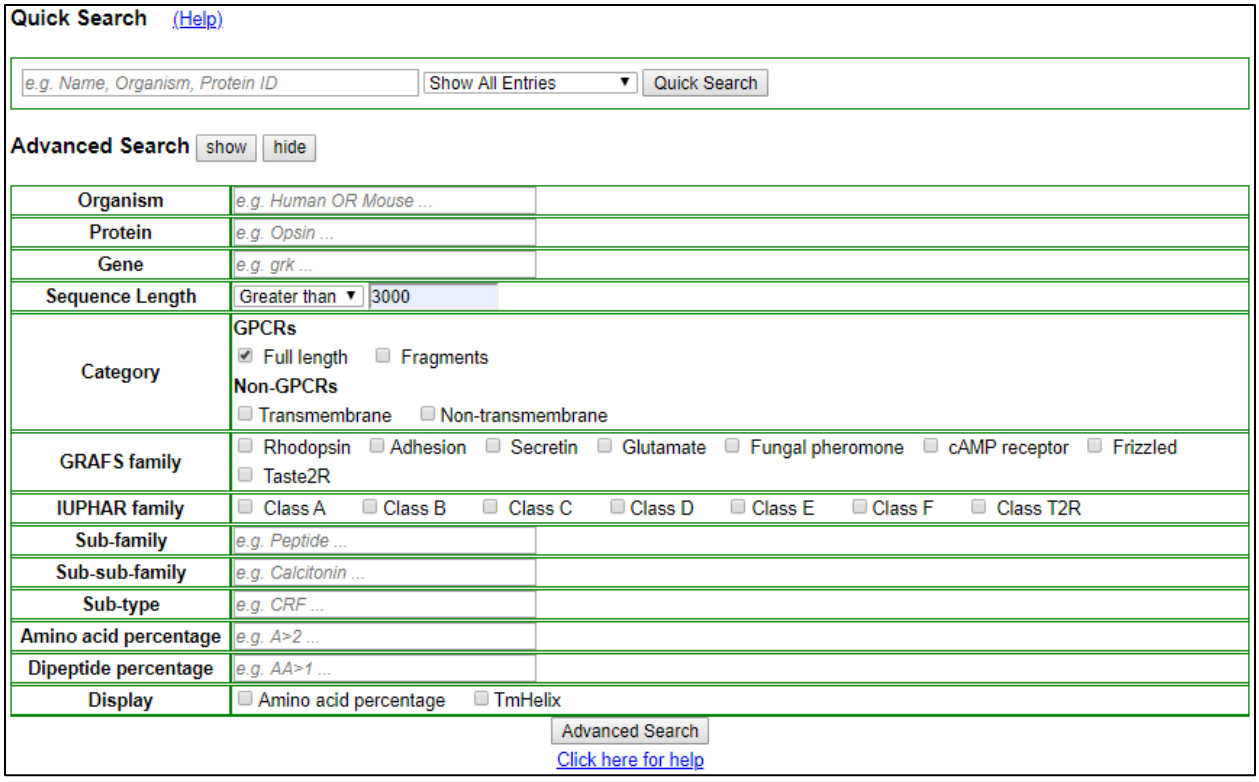


Figure 5.2: Web-interface of GPCR-PEnDB

For example, if we use the advanced search option to look for GPCR sequences with lengths greater than 3000, the result (see Figure 5.3) shows that there are 10 confirmed GPCR sequences, from different organisms such as human, mouse, rat, fruitfly and zebra fish. All these sequences belong to Class B (Secretin-like/Adhesion-like family). This tells us that GPCRs from other families do not exceed 3000 in length. Also, if we choose the option to display the available TMHMM predicted regional lengths, we can see that the N-terminals are much longer (~2400 to ~5900) than the C-terminals (~150 to ~300) in these long Class B GPCRs.

Results Table

The number of proteins found: 10
Query terms: Confirmed GPCRs + Length > 3000
Display: GRAFS & IUPHAR
Download: Result table(csv) FASTA file

Search

ID	Entry name	Protein name	Gene	Length	Type	Genus	Species	Common name	GRAFS (family)	IUPHAR (family)	Sub-family	Sub-sub-family	Sub-type	PDB_ID
Q9V5N8	STAN_DROME	Protocadherin-like wing polarity protein ...	stan	3579	CG	Drosophila	melanogaster	Fruit fly	Secretin-like	Class B	-	-	-	-
Q8IZF6	AGRG4_HUMAN	Adhesion G-protein coupled receptor G4	ADGRG4	3080	CG	Homo	sapiens	Human	Adhesion-like	Class B	Adhesion receptors	ADGRG	ADGRG4	-
Q8WXG9	GPR98_HUMAN	Adhesion G-protein coupled receptor V1	ADGRV1	6306	CG	Homo	sapiens	Human	Adhesion-like	Class B	Adhesion receptors	ADGRV	ADGRV1	-
Q9NYQ6	CELR1_HUMAN	Cadherin EGF LAG seven-pass G-type recep ...	CELSR1	3014	CG	Homo	sapiens	Human	Adhesion-like	Class B	Adhesion receptors	ADGRC	*CELSR1*	-
Q9NYQ7	CELR3_HUMAN	Cadherin EGF LAG seven-pass G-type recep ...	CELSR3	3312	CG	Homo	sapiens	Human	Adhesion-like	Class B	Adhesion receptors	ADGRC	*CELSR3*	-
B7ZCC9	AGRG4_MOUSE	Adhesion G-protein coupled receptor G4	Adgrg4	3073	CG	Mus	musculus	Mouse	Adhesion-like	Class B	Adhesion receptors	ADGRG	ADGRG4	-
O35161	CELR1_MOUSE	Cadherin EGF LAG seven-pass G-type recep ...	Celsr1	3034	CG	Mus	musculus	Mouse	Adhesion-like	Class B	Adhesion receptors	ADGRC	*CELSR1*	-
Q91ZI0	CELR3_MOUSE	Cadherin EGF LAG seven-pass G-type recep ...	Celsr3	3301	CG	Mus	musculus	Mouse	Adhesion-like	Class B	Adhesion receptors	ADGRC	*CELSR3*	-
Q6JAN0	GPR98_DANRE	Adhesion G-protein coupled receptor V1	adgrv1	6199	CG	Danio	rerio	Zebra fish	Adhesion-like	Class B	Adhesion receptors	ADGRV	ADGRV1	-
O88278	CELR3_RAT	Cadherin EGF LAG seven-pass G-type	Celsr3	3313	CG	Rattus	norvegicus	Rat	Adhesion-like	Class B	Adhesion receptors	ADGRC	*CELSR3*	-

Figure 5.3: Resulting page showing the GPCR sequences with >3000 length.

Chapter 6: Assessment of existing webservers

Using the entire GPCR-PEnDB dataset, an assessment is performed on several webservers that are available online. At first it was done on predicting a GPCR and distinguishing a non-GPCR from the given dataset, the transmembrane non-GPCRs are also kept separated from the non-transmembrane non-GPCRs in order to analyze the false positive rates separately. Then for the webservers that can predict other levels of classification, it was done using the true positive GPCRs. As Classes A, B and C are larger families than the rest, we analyzed only using these classes for the family level. The following statistical measures are used to conduct the assessment:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total}}$$

$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}}$$

$$\text{Positive Predictive Value, PPV} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Negative Predictive Value, NPV} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Negative}}$$

$$\text{False Positive Rate, FPR} = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}}$$

The assessment was challenging due to the requirements of the input data for each web-server and also parsing the result from the downloadable files or the resulting html page. Some of them only allows to input one sequence at a time or cannot perform when a special character (not representing the usual amino acids) are present in a sequence. The following section focuses on the whole process of the assessment done on each of the webservers.

GPCR-CA

This web-server (Xiao et al., 2009) is based on a cellular automaton (CA) approach described by Wolfram (1984) where the CA images are used for the complicated and long protein sequences to observe the distinctive pattern features even without the presence of significant homology. At first the 20 amino acids are coded in Binary digits (e.g. Proline, P = 00001; Arginine, R = 00110 etc.) based on the model that uses similarity and complementary rule along with recognition and information theory. For a given protein sequence and based on the amino acid residues organized using five-digit binary numbers, the grids are filled with the colors white and black if the binary digit for a position is 0 or 1 respectively. For each protein sequences this procedure creates narrow ribbons filled with black and white colors that can be further analyzed to recognize the textural patterns among the given set of data. To optimize and extract the features from these patterns, gray level co-occurrence matrix (GLCM) is used from which four distinct features are selected, namely angular second moment, contrast, inverse different moment and entropy. These four features are then added as components in the Pseudo amino acid composition (Chou, 2001) and by using the Covariant Discriminant algorithm the prediction is completed. This web-server can predict and classify GPCRs up to the family level. The families include Class A, B, C, D, E and F. This web-server only allows one sequence at a time as an input, therefore, a Python module, “Mechanize” is used that allows user to programmatically automate and parse data by filling the forms in the HTML page. Following Figure 6.1 shows the output page for a given input from which the input is parsed (Appendix L) for every protein query.

Figure 6.1: GPCR-CA html page source for a given input

The overall accuracy for predicting a GPCR and a non-GPCR for this web-server is 88.68% is because of the false positive rate for transmembrane proteins is 27.92%, whereas the server performs well classifying the GPCR families with an accuracy of 91.30%.

PCA-GPCR

This web-server [Peng et al., 2010] uses principal component analysis (PCA) to reduce the high dimensionality of the data by converting a set of observations of correlated variables to a set of uncorrelated variables named principal components. Here several physiochemical properties are used as feature variables along with some autocorrelation descriptors which led the number of variables to be 1497. PCA is used for the dimensionality reduction and an intimate sorting algorithm is used afterwards to predict different classes of GPCRs.

For a given data matrix, $\mathrm{X} = \left(\mathrm{X}_1,\ \mathrm{X}_2, \ldots \ldots \mathrm{X}_\mathrm{p}\right)$ with m number of feature variables where X_i is the column vector where $\mathrm{i} = 1, 2, \ldots \ldots, \mathrm{n}$ denoting all the protein sequences. Using the mean and variance of X_i the data is standardized as follows

$$\mathrm{Y_i} = \frac{\mathrm{X_i} - \overline{\mathrm{X}}_\mathrm{i}}{\sqrt{\mathrm{Var(X_i)}}} \text{ where } \mathrm{i} = 1,2,\ldots,\mathrm{p}$$

Then the covariance matrix, $\mathrm{Cov(Y)}$ is determined by computing

$$\mathrm{Cov(Y)} = \frac{1}{n-1}\mathrm{Y^T Y}$$

Then by calculating the eigenvalues $(\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p)$ and the eigenvectors $(E_1, E_2, \dots, E_p)$ for the covariance matrix $\mathrm{Cov(Y)}$ the construction of the principal components is done where only a first number of m uncorrelated PCs are used where $\mathrm{m} \leq \mathrm{p}$ representing the protein sequences.

$$PC(i) = \sum_{j=1}^{p} E_{i,j} Y_j \text{ where i} = 1,2, \dots, \mathrm{p}$$

The principal components obtained from a given set of data are then trained by the intimate sorting algorithm where based on the similarity score a query protein is grouped with the trained data. Given a set of N training proteins P_i $(i = 1,2, \dots, N)$ with a λ dimension of feature vector, for a given query protein with the same number of feature variables the similarity score is calculated by the following

$$\phi(\mathrm{P}, \mathrm{P_i}) = \frac{\mathrm{P} \cdot \mathrm{P_i}}{\|\mathrm{P}\| \|\mathrm{P_i}\|} \text{ where i} = 1,2, \dots, \mathrm{N}$$

$$\text{Also, } \mathrm{P} \cdot \mathrm{P_i} = \sum_{\mathrm{j}=1}^{\lambda} \mathrm{p_j p_{i,j}}\ ,\ \|\mathrm{P}\| = \sqrt{\sum_{\mathrm{j}=1}^{\lambda} \mathrm{p_j^2}} \text{ and } \|\mathrm{P_i}\| = \sqrt{\sum_{\mathrm{j}=1}^{\lambda} \mathrm{p_{i,j}^2}}$$

Here, the similarity score, $\phi(\mathrm{P}, \mathrm{P_i})$ for a query protein varies from -1 to 1, where the higher score results a query protein being grouped in a class based on the training data. Also, the highest accuracy is achieved in this study when the number of PCs is chosen to be 32.

As PCA-GPCR is created in 2010 and follows the hierarchical structure provided by the GPCRdb which at that time divided GPCRs into five main classes, namely Rhodopsin (Class A), Secretin (Class B), Glutamate (Class C), Vomeronasal and Taste 2 receptors. Therefore, this web-server can classify only those classes and some of the sub-families, sub-sub-families and sub-types of Class A GPCRs. This web-server had an overall low performance due to it being a legacy program that followed a different set of proteins and their classification system. Although it had a higher Sensitivity (99.62%) than any other ones on the assessment.

SVM-PROT

This web-based software [Y. H. Li et al., 2016] uses support vector machine (SVM), a supervised learning algorithm where the data is labeled and trained for identifying an optimal hyperplane or multiple hyperplanes in order to classify the new (testing data). A hyperplane is a line that separates samples and groups them based on their properties or feature variables. Then margins are used to define the minimal distance for a sample from the hyperplane that can be used to minimize the complexity for separating the classes from each other. The samples on the margins are known as the support vectors that helps to determine the safe distance from which the classes can be separated. In a two-dimensional space, it is easier to separate two classes using the hyperplane and margins on both sides if the samples from the two classes are very distinct from each other. But for a complicated set of data, where it is not easily separable, more dimensions (e.g. z-axis for three-dimension) can be added for separating the classes. For example, when z-axis is needed to separate the classes, the distance of the data points is calculated from the z-origin and mapped back to get the boundary for separating the classes. This method of solving a non-linear problem using a linear classifier is known as the kernel.

The feature vector consists of the physicochemical properties generated from the protein sequences. The properties include amino acid composition, hydrophobicity, normalized Van der Waals volume, polarity, polarizability, charge, surface tension, secondary structure and solvent accessibility for the residues of a sequence. Then three global descriptors are used to describe these properties where the percentage of amino acids, counts of transition from one amino acid to the other and amino acid percentages in the 25, 50, 75 and 100% of the sequence length are measured.

For constructing the hyperplane, the feature vector for each sequence is used to find the vector normal to the hyperplane (w) and minimizing that vector (Euclidean norm of w, $\|w\|^2$) by using another variable (b) that satisfies the following two conditions where y_i is the group index and $|b|/\|w\|$ represents the perpendicular distance from the hyperplane to the origin.

$w \cdot x_i + b \geq +1, \text{for } y_i = +1 \text{ (positive) Group 1}$

$w \cdot x_i + b \leq -1, \text{for } y_i = -1 \text{ (negative) Group 2}$

The sign after determining $[w \cdot x + b]$ classifies the given vector x. This procedure is used for linearly separating two classes. For non-linearly separable classes, SVM-PROT uses a high dimensional space for mapping all the input variables using a Gaussian kernel function and a Lagrangian expression.

Gaussian kernel, $K(x_i, x_j) = e^{-\|x_j - x_i\|^2 / 2\sigma^2}$

Lagrangian: $\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \, K(x_i, x_j)$ where $\alpha_i \geq 0 \; and \; \sum_{i=1}^{l} \alpha_i y_i = 0$

Here, l is the number of training samples, α is a vector of l variables and each training sample (x_i, y_i) is represented by α_i.

By maximizing the Lagrangian expression α_i^0 and b are determined for applying them in the following decision function that places the input variable in the high dimensional feature space whether it belongs to a positive or a negative group.

$f(x) = \text{sign}\left(\sum_{i=1}^{l} \alpha_i^0 \, y_i K(x_i, x_j) + b\right)$

SVM-PROT was published in 2003 and now it covers a wide range of protein families (192) that can be predicted and not only limited to predicting GPCRs also defining the molecular and biological function using the machine learning method. The program only allows input sequences of the length starting from 50 to 5000 amino acids. If a large dataset is given where at least one of the sequences has a special/unidentified amino acid, it pauses for the entire dataset and is unable to produce the output file. Keeping that in mind, the number of sequences from our original dataset is reduced to 3063 GPCRs (including fragments), 1315 non-GPCRs and 2219 transmembrane non-GPCRs.

The complete dataset is separated for GPCRs, transmembrane non-GPCRs and non-GPCRs to save time for getting the output files. The output file is in .xls format that provides ID, Length and three functions namely, Molecular function with probability (in percentage), Biological process and Broadly defined function. Sometimes the output files do not follow the

pattern of having all the three functions or even not having any, so each possible case has been considered in the script. At first, the strategy to parse the output for the GPCR dataset is discussed.

To get the prediction result, the molecular function needs to be parsed along with the probabilities for each protein. As it is a general protein family prediction web-server, the top predicted families with a higher probability for a confirmed GPCR can be found as either "G protein coupled receptors" or "7 transmembrane receptor (rhodopsin family)" with a value higher than 90%. Therefore, the cut-off value is set to be greater than or equal to 90% to consider a protein predicted as a GPCR.

Generally, if it is not predicted as a GPCR, the top hit or probability is usually 58.6%. Figure 6.2 (left) shows an example that is considered in this assessment as a GPCR and the figure 6.2 (right) is considered as a predicted non-GPCR. After looking at the file in the excel sheet, the script is written in such a way where it considers the first top three hits and their corresponding probabilities.

	A	B	C	D
1	Query=O93441			
2				
3	Length=354 amino acids			
4				
5	Your protein may belong to the following families:			
6				
7	Protein Family Name	GO Category	Probability(%)	
8				
9	Molecular Function:			
10				
11	7 transmembrane receptor (rhodopsin family and chemorecept	GO:0004930	99.2	
12	G protein coupled receptors	GO:0004930	99.1	
13	Photoreceptor	GO:0009881	99	
14	All lipid-binding proteins	GO:0008289	98.6	
15	TC2.A Electrochemical Potential-driven transporters - Porters (u	GO:0015291	98	
16	Zinc-binding	GO:0008270	76.2	
17	Sodium-binding	GO:0031402	73.8	
18	Metal-binding	GO:0046872	62.2	
19	7 transmembrane receptor (secretin family)	GO:0004930	58.6	
20				
21	Biological Process:			
22				
23	Chromophore	GO:0018298	98.7	
24				
25	Broadly Defined Function:			
26				
27	Transmembrane	GO:0016021	98.2	

	A	B	C	D
170	Query=G5ECX0			
171				
172	Length=1338 amino acids			
173				
174	Your protein may belong to the following families:			
175				
176	Protein Family Name	GO Category	Probability(%	
177				
178	Molecular Function:			
179				
180	Zinc-binding	GO:0008270	80.4	
181	Metal-binding	GO:0046872	62.2	
182	7 transmembrane receptor (secretin family)	GO:0004930	58.6	
183				
184	Biological Process:			
185				
186	Chlorophyll biosynthesis	GO:0015995	58.6	
187	DNA repair	GO:0006281	58.6	
188				
189	Broadly Defined Function:			
190				
191	Transmembrane	GO:0016021	98.1	

Figure 6.2: Examples of output from SVM-PROT

Also, it is observed that this web-server can only classify the GPCR families Class A, B and C. Therefore, the hits that start with "7 transmembrane receptor" are also parsed carefully to get the family level prediction result. For the non-GPCRs some of the proteins got predicted with no molecular function or probabilities, in those cases the proteins are still considered as non-

GPCRs as this web-server is a general protein prediction tool. After parsing the data another script is used to compare the results with our dataset and the statistical calculations are done using Microsoft excel.

Although it is not an exclusive web-server for predicting only GPCRs, the web-server is very efficient for prediction of a GPCR with an accuracy, sensitivity and specificity being higher than 95% and the false positive value is also very low (5.67%). For family level classification, the predicted output is measured only for Class A, B and C resulting a 95.05% accuracy.

Table 6.1: Assessment on the available web-servers (Part I)

	PCA-GPCR	**GPCR-CA**	**SVM-PROT**
Accuracy	72.90	88.68	96.57
Sensitivity	99.62	96.38	96.57
Specificity	51.60	81.97	96.56
Positive Predicted Value	62.14	82.44	95.77
Negative Predicted Value	99.42	96.27	97.22
False Positive Value-TM	70.64	27.92	5.76
Family Level Accuracy	-	91.30	95.05

The above Table 6.1 shows the statistical measures (in percentage) calculated using the web-servers that are available online from which PCA-GPCR can only predict GPCRs and GPCR-CA can predict and classify GPCRs to six families (excluding Taste 2 receptors). SVM-PROT is a web-based software that is used to predict and classify protein families and out-performed than the other GPCR predicting web-servers as it is trained using a wide range of protein families.

The following three web-servers are also available online but we have locally installed and implemented them in our GPCR-PEn pipeline to obtain a well-formatted result table and calculate the statistical measures for observing their strengths and weaknesses individually as well as from generating an ensemble approach.

GPCR-Pred

This web-server [Bhasin & Raghava, 2004] also uses support vector machine (SVM) based algorithm in a three-step strategy where the fixed-length feature vector is obtained from the dipeptide compositions of the protein sequences. SVM_light [Joachims, 1998] software package is implemented in this server where number of parameters and built-in kernel functions can be specified and used.

GPCR-Pred can predict the five families of GPCRs except Frizzled (Class F) and only the list of sub-families of Class A collected from GPCRdb. This SVM based module has been developed using 778 GPCRs and 2524 non-GPCRs. As this web-server is created in 2004, the sub-families of Class A do not match with the latest classification system of GPCRdb. In the web-server the output cannot be downloaded and only displayed in a resulting HTML page. Therefore, to display and save the result for multiple sequences, the local version of GPCRPred is used in our pipeline. A function is created in the GPCR-PEn pipeline that searches for the word "Class" or "Subfamily" to parse the result for each sequence from a given fasta formatted file of multiple sequences.

Results Table

Download: Results / Fasta_Submission

Search

Index	Protein	Prediction Type	Prediction Count	Length (AA)	GPCRPred Class	GPCRPred Subclass	Sequence
G_1	sp|O93441|OPSD_GALML	Full Length GPCR	1	354	Class A	Rhodopsin	mngtegenfyvpmsnktgvvrnpfeypqyyladhwmfavl ...
G_3	sp|A0A0K3AWM6|MOM5_CAEEL	Full Length GPCR	1	570	Class A	Peptide	mhrhililflfgclsadqrlsstsissmngfsttrkcehi ...
G_4	sp|G5ECB2|GABR2_CAEEL	Full Length GPCR	1	842	Class A	Peptide	msrwlslllfavqavggyeageelsckrrhggiplplgvf ...
G_5	sp|G5ECD9|AEX2_CAEEL	Full Length GPCR	1	321	Class A	Lysospingolipid	mnstdiianvtkpfvenltlgetafyiscgivgtvfnalv ...
G_6	sp|G5ECQ2|FRIZ2_CAEEL	Full Length GPCR	1	578	Class A	Peptide	mllrisvlfllgscgalfgkrqkceqitiplckgigynm ...
G_7	sp|G5EDW2|LPLT1_CAEEL	Full Length GPCR	1	1014	Class-B	-	mrrnkttysllqtilvaclltvtptfasnkpttdesgtis ...

Figure 6.3: Output table of GPCRPred in the GPCR-PEn pipeline

The output table only shows the proteins that got predicted as GPCRs, for the non-GPCRs no output is shown, so for the non-GPCRs that is not displayed or in the table, are considered as non-GPCRs for the statistical calculations. The accuracy is found to be less than 90% and the reason behind that is the false positive rate using the transmembrane non-GPCRs gave comparatively a very high (36.83%) value.

Pfam

Profile hidden Markov models (HMM) are probabilistic models and here it is applied in the multiple sequence alignment to identify the position specific conserved amino acids in the sequences to enclose the evolutionary changes that has occurred [Finn et al., 2016]. Initially from the multiple sequence alignments, the insertion/deletion of amino acid residues are modelled which leads to estimate the frequency of the amino acids at each position and observing the transitions of the amino acids for each specific position from the alignments for creating the profile HMM. Then this profile is used to find addition matching family members from the reference proteome database that qualifies the insertion threshold value.

This is a general protein prediction tool that is not only exclusive for predicting GPCRs. Therefore, the name of the families of GPCRs have different notations (Table 6.2) than the traditional manner. For convenience and interpreting the names used in Pfam, it is also used locally from our pipeline where a list of names has been collected to accurately get the prediction accuracy for the given set of data. This web-server can predict the classes A, B, C, E, F and Taste2 receptors, also five sub-families of Class A like olfactory and chemoreceptors etc.

Table 6.2: The Pfam notations for different classes of GPCRs

Pfam notations	Receptor name	Class (Family)
7tm_1	Rhodopsin	Class A
7tm_4	Olfactory	
V1R	Vomeronasal 1	
7tm_2	Secretin	Class B
7tm_3	Metabotropic glutamate	Class C
Dicty_CAR	cAMP	Class E

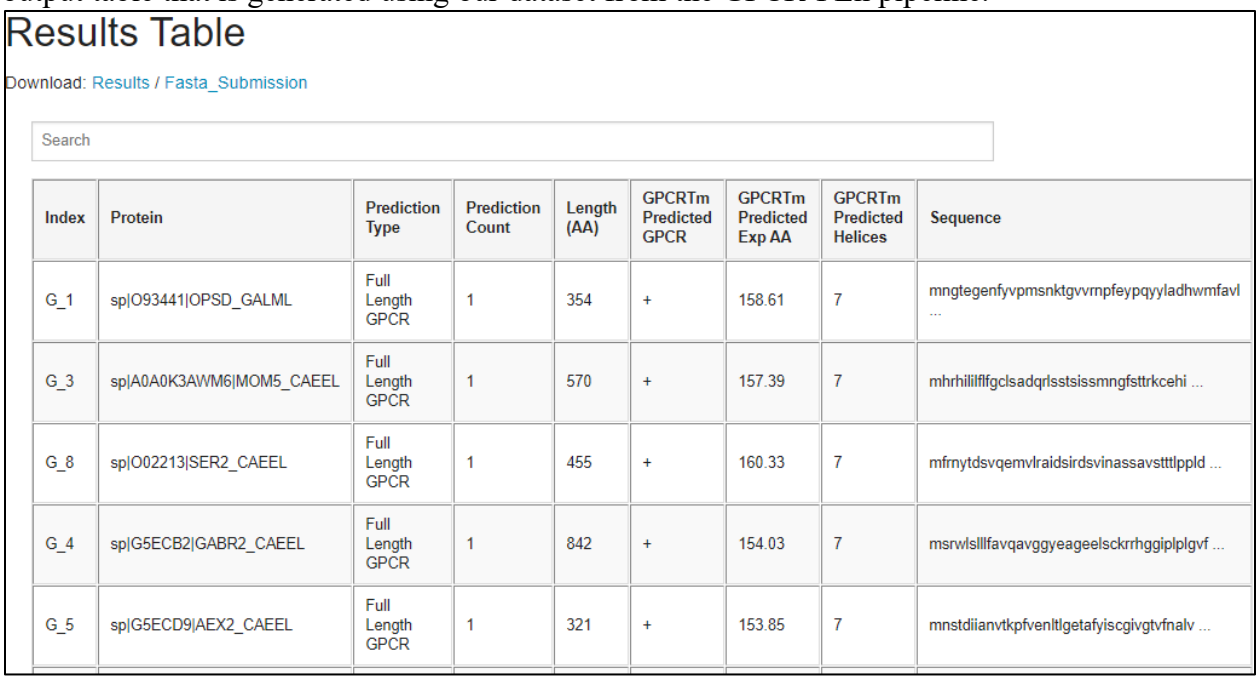
Frizzled	Frizzled	Class F
TAS2R	Taste 2	Class T2R

The overall performance of Pfam is very good except the Sensitivity is lower compared to the other web-servers, but this web-server gave the best false positive value which is 0.45%.

GPCR-TM

The local version of TMHMM 2.0 [Sonnhammer et al., 1998] has been implemented in our pipeline as GPCR-TM that predicts a transmembrane protein using Hidden Markov model (HMM) and constructing a model considering the residues from different regions of a protein structure. The core and caps of the transmembrane helices along with the loops are the three main locations that is used to build the model. The cytoplasmic and non-cytoplasmic sides of a protein carry different distribution of the residues, therefore, in total the model has seven states. The first one is the helix core, in the cytoplasmic side it consists of a cap, a loop and a globular domain, whereas in the non-cytoplasmic side it contains a cap, a short and a long loop with a globular domain. In all the states the emission probabilities of the amino acids are estimated. For the training of the HMM model, at first the maximum likelihood is estimated then re-estimated from the labeled sequences obtained from Swiss-Prot to add soft boundaries. Then an annealing scheme is used where, at first noise is added to the model to get the unfavorable models sampled with some probability and with every iteration the noise is reduced by 5%. Which basically unlabeled the residues by a certain amount for all the boundaries and then relabeled the residues according to the predicted result. After the convergence, using the Viterbi algorithm the most likely path of the sample sequence is determined. This method is repeated where the relabeled data from the previous state is used on the new data to be unlabeled and then relabeled. This HMM-based method specializes on predicting the different regional lengths of a transmembrane protein that we have used here to predict the GPCRs which is also known as the seven transmembrane proteins. Several conditions have been set to make this tool fit for predicting GPCRs through our pipeline, such as the length of the protein sequences must be higher than 150 and only the sequences that predicted

with seven transmembrane helices can be considered as predicted GPCRs. Following shows the output table that is generated using our dataset from the GPCR-PEn pipeline.



Results Table

Download: Results / Fasta_Submission

Search

Index	Protein	Prediction Type	Prediction Count	Length (AA)	GPCRTm Predicted GPCR	GPCRTm Predicted Exp AA	GPCRTm Predicted Helices	Sequence
G_1	sp|O93441|OPSD_GALML	Full Length GPCR	1	354	+	158.61	7	mngtegenfyvpmsnktgvvrnpfeypqyyladhwmfavl ...
G_3	sp|A0A0K3AWM6|MOM5_CAEEL	Full Length GPCR	1	570	+	157.39	7	mhrhililflfgclsadqrlsstsissmngfsttrkcehi ...
G_8	sp|O02213|SER2_CAEEL	Full Length GPCR	1	455	+	160.33	7	mfrnytdsvqemvlraidsirdsvinassavstttlppld ...
G_4	sp|G5ECB2|GABR2_CAEEL	Full Length GPCR	1	842	+	154.03	7	msrwlslllfavqavggyeageelsckrrhggiplplgvf ...
G_5	sp|G5ECD9|AEX2_CAEEL	Full Length GPCR	1	321	+	153.85	7	mnstdiianvtkpfvenltlgetafyiscgivgtvfnalv ...

Figure 6.4: Result table of GPCR-Tm using GPCR-PEn pipeline

As this tool only predicts the regional lengths and highly dependent on predicting a GPCR by calculating the number of transmembrane helices, it also had a higher false positive rate but the overall performance is much more alike with GPCRPred.

Table 6.3 shows the assessment (in percentage) done on the tools individually and altogether that are ensembled in the GPCR-PEn pipeline. The ∪ (union) of three tools represents when any of these predicting a protein to be a GPCR or a non-GPCR which shows that the sensitivity, measuring the true positive rate is higher and the specificity, measuring the true negative rate is lower. In case of ∩ (intersection), a protein is predicted as a GPCR (true positive) or a non-GPCR (true negative) when all the tools are predicting the same for a protein. This shows that the sensitivity is much lower and the specificity is higher. It is also observed that the accuracy is lower for the union of the tools and the false positive rate is really low for the intersection of the tools.

Table 6.3: Assessment summary on the web-servers (Part II)

	Pfam	**GPCRPred**	**GPCR-Tm**	**Three tools ∪**	**Three tools ∩**
Accuracy	91.17	86.32	88.76	82.29	89.30
Sensitivity	80.02	97.98	95.82	99.77	77.15
Specificity	99.72	76.95	83.09	67.24	99.75
Positive Predicted Value	99.57	77.35	81.99	72.38	99.62
Negative Predicted Value	86.13	97.93	96.11	99.71	83.53
False Positive Rate -TM	0.45	36.83	26.98	32.76	0.25
Family Level Accuracy	99.96	95.10	-	-	-

It is observed that the general protein databases that are not focused on only one type of protein families did better than the others as the dataset used for those programs are much larger and full of variation. The programs tend to predict any transmembrane protein to be a GPCR, hence, for the negative dataset it is very important to train the algorithms with more transmembrane proteins. Some of the programs followed older version of classification that is not followed anymore caused a lower score in the assessment. Also, due to the lack of data in different lower level classification groups the programs were unable to perform well on those levels and more data needs to be incorporated.

Chapter 7: Conclusion and Future Work

7.1 Conclusion

GPCR-PEnDB is useful for quickly identifying one or multiple GPCR proteins from different species or based on their families and subfamilies. The way the basic features are stored within the database make it a powerful tool for generating datasets for creating and testing algorithms and tools related to GPCR prediction and classification. The GPCR-PEnDB database serves a cross disciplinary purpose so that any user who needs a particular set of information about GPCR sequences can easily obtain it.

7.2 Future Work

The aims of this research focused on setting up a database that can be used for assessing the performances of various programs for predicting and classifying GPCRs. Through this project I have identified a few directions in which GPCR-PEnDB can be further extended, refined, and utilized.

Prediction

The computational tools that have been used to predict a novel GPCR, are mostly trained and tested on human or other model organism data. We have not only incorporated confirmed GPCR data from 410 organisms but have covered more than 800 different organisms in the negative (non-GPCR) example set as well. This negative set has less than 50% sequence similarity and also integrates more examples of the transmembrane proteins. Using a combination of these dataset researchers will find it useful to evaluate the strengths and weaknesses of their algorithms when applied to a more diverse range of organisms.

Classification

GPCR-PEnDB contains 3129 confirmed GPCRs, where the classification information up to level five is only available for ~64% of the data. To enhance the availability of more classification levels for the existing GPCRs as well as newly predicted ones, several approaches can be incorporated. Studying the three-dimensional structures of the confirmed and predicted GPCRs and analyzing the differences between these two types will be useful to understand the GPCR-ligand binding modes. Keeping that in mind the related information on the ligands and the G proteins for the available data can be incorporated as well.

While this initial version has been implemented at this time, GPCR-PEnDB provides the necessary framework for growth as more data sequences and features can be included in future developments. Ongoing work to expand the sequences within the database include adding a collection of predicted but not yet confirmed GPCRs and revising the non-GPCR sequence collection that better resembles the organism distribution of the GPCRs.

Additionally, the types of information found within the database can be expanded. For example, to help understand the ligand binding and signaling in GPCRs. it is very important to observe the 3D structure of a particular GPCR, with the overall available structures in hand. We plan to develop a pattern specifically focused on ligand binding and the internal signaling inside of a cell developed by a specific type of GPCRs as well as classifying them based on their behavior and how they work with the agonists and antagonists. Incorporating available GO-terms could also be very useful in grouping each GPCR based on their biological process or molecular function to understand the mechanism of each individual GPCR.

References

Alexander, S. P., Christopoulos, A., Davenport, A. P., Kelly, E., Marrion, N. V, Peters, J. A., … CGTP Collaborators. (2017). THE CONCISE GUIDE TO PHARMACOLOGY 2017/18: G protein-coupled receptors. *British Journal of Pharmacology*, *174 Suppl 1*, S17–S129. https://doi.org/10.1111/bph.13878

Alexander, S. P., Davenport, A. P., Kelly, E., Marrion, N., Peters, J. A., Benson, H. E., … Collaborators, C. (2015). The Concise Guide to PHARMACOLOGY 2015/16: G protein-coupled receptors; The Concise Guide to PHARMACOLOGY 2015/16: G protein-coupled receptors. *British Journal of Pharmacology*, *172*, 5744–5869. https://doi.org/10.1111/bph.13348/full

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Bairoch, A., & Apweiler, R. (1997). *The SWISS-PROT protein sequence data bank and its supplement TrEMBL*. *Nucleic Acids Research* (Vol. 25).

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., … Bourne, P. E. (2000). The protein data bank. *Nucleic Acids Research*, *28*(1), 235–242. https://doi.org/10.1093/nar/28.1.235

Bhasin, M., & Raghava, G. P. S. (2004). GPCRpred: An SVM-based method for prediction of families and subfamilies of G-protein coupled receptors. *Nucleic Acids Research*, *32*(WEB SERVER ISS.), 383–389. https://doi.org/10.1093/nar/gkh416

Bruno, A., Costantino, G., de Fabritiis, G., Pastor, M., & Selent, J. (2012). Membrane-sensitive conformational states of helix 8 in the metabotropic Glu2 receptor, a class C GPCR. *PLoS ONE*, *7*(8). https://doi.org/10.1371/journal.pone.0042023

Chan, W. K. B., Zhang, H., Yang, J., Brender, J. R., Hur, J., Ozgur, A., & Zhang, Y. (2015). GLASS: A comprehensive database for experimentally validated GPCR-ligand associations. *Bioinformatics*, *31*(18), 3035–3042. https://doi.org/10.1093/bioinformatics/btv302

Chaudhari, N., Landin, A. M., & Roper, S. D. (2000). A metabotropic glutamate receptor variant functions as a taste receptor. *Nature Neuroscience*, *3*(2), 113–119. https://doi.org/10.1038/72053

Chou, K.-C. (2001). Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins: Structure, Function, and Genetics*, *43*(3), 246–255. https://doi.org/10.1002/prot.1035

Chou, K.-C., & Elrod, D. W. (1999). Protein subcellular location prediction. *Protein Engineering, Design and Selection*, *12*(2), 107–118. https://doi.org/10.1093/protein/12.2.107

Cobanoglu, M. C., Saygin, Y., & Sezerman, U. (2011). Classification of GPCRs using family specific motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *8*(6), 1495–1508. https://doi.org/10.1109/TCBB.2010.101

Davies, M. N., Secker, A., Freitas, A. A., Mendao, M., Timmis, J., & Flower, D. R. (2007). On the hierarchical classification of G protein-coupled receptors. *Bioinformatics*, *23*(23), 3113–3118. https://doi.org/10.1093/bioinformatics/btm506

Davies, M. N., Secker, A., Halling-Brown, M., Moss, D. S., Freitas, A. a, Timmis, J., … Flower, D. R. (2008). GPCRTree: online hierarchical classification of GPCR function. *BMC*

Research Notes, *1*, 67. https://doi.org/10.1186/1756-0500-1-67
de Graaf, C., Song, G., Cao, C., Zhao, Q., Wang, M. W., Wu, B., & Stevens, R. C. (2017, December 1). Extending the Structural View of Class B GPCRs. *Trends in Biochemical Sciences*. Elsevier Ltd. https://doi.org/10.1016/j.tibs.2017.10.003
Edgar, R. C. (2010). Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, *26*(19), 2460–2461. https://doi.org/10.1093/bioinformatics/btq461
Eilers, M., Hornak, V., Smith, S. O., & Konopka, J. B. (2005). Comparison of class A and D G protein-coupled receptors: common features in structure and activation. *Biochemistry*, *44*(25), 8959–8975. https://doi.org/10.1021/bi047316u
Elrod, D. W., & Chou, K. C. (2002). A study on the correlation of G-protein-coupled receptor types with amino acid composition. *Protein Eng*, *15*(9), 713–715.
Finn, R. D., Coggill, P., Eberhardt, R. Y., Eddy, S. R., Mistry, J., Mitchell, A. L., … Bateman, A. (2016). The Pfam protein families database: towards a more sustainable future. *Nucleic Acids Research*, *44*(D1), D279–D285. https://doi.org/10.1093/nar/gkv1344
Gao, Q. Bin, & Wang, Z. Z. (2006). Classification of G-protein coupled receptors at four levels. *Protein Engineering, Design and Selection*, *19*(11), 511–516. https://doi.org/10.1093/protein/gzl038
Gentry, P. R., Sexton, P. M., & Christopoulos, A. (2015). Novel Allosteric Modulators of G Protein-coupled Receptors. *The Journal of Biological Chemistry*, *290*(32), 19478–19488. https://doi.org/10.1074/jbc.R115.662759
Gookin, T. E., Kim, J., & Assmann, S. M. (2008). Whole proteome identification of plant candidate G-protein coupled receptors in Arabidopsis, rice, and poplar: computational prediction and in-vivo protein coupling. *Genome Biology*, *9*(7), R120. https://doi.org/10.1186/gb-2008-9-7-r120
Guerrero, F. D., Kellogg, A., Ogrey, A. N., Heekin, A. M., Barrero, R., Bellgard, M. I., … Leung, M.-Y. (2016). Prediction of G protein-coupled receptor encoding sequences from the synganglion transcriptome of the cattle tick, Rhipicephalus microplus. *Ticks and Tick-Borne Diseases*, *7*(5), 670–677. https://doi.org/10.1016/j.ttbdis.2016.02.014
Guo, Y. Z., Li, M., Lu, M., Wen, Z., Wang, K., Li, G., & Wu, J. (2006). Classifying G protein-coupled receptors and nuclear receptors on the basis of protein power spectrum from fast Fourier transform. *Amino Acids*, *30*(4), 397–402. https://doi.org/10.1007/s00726-006-0332-z
Hamann, J., Aust, G., Arac, D., Engel, F. B., Formstone, C., Fredriksson, R., … Schioth, H. B. (2015). International Union of Basic and Clinical Pharmacology. XCIV. Adhesion G Protein-Coupled Receptors. *Pharmacological Reviews*, *67*(2), 338–367. https://doi.org/10.1124/pr.114.009647
Hirai, T., Subramaniam, S., & Lanyi, J. K. (2009, August). Structural snapshots of conformational changes in a seven-helix membrane protein: lessons from bacteriorhodopsin. *Current Opinion in Structural Biology*. https://doi.org/10.1016/j.sbi.2009.07.009
Horn, F. (2003). GPCRDB information system for G protein-coupled receptors. *Nucleic Acids Research*, *31*(1), 294–297. https://doi.org/10.1093/nar/gkg103
Horn, F., Vriend, G., & Cohen, F. E. (2001). Collecting and harvesting biological data: the GPCRDB and NucleaRDB information systems. *Nucleic Acids Research*, *29*(1), 346–349. https://doi.org/10.1093/nar/29.1.346
Horn, F., Weare, J., Beukers, M. W., Hörsch, S., Bairoch, A., Chen, W., … Vriend, G. (1998).

GPCRDB: an information system for G protein-coupled receptors. *Nucleic Acids Research*, *26*(1), 275–279. https://doi.org/10.1093/nar/26.1.275

Hu, G.-M., Mai, T.-L., & Chen, C.-M. (2017). Visualizing the GPCR Network: Classification and Evolution. *Scientific Reports*, *7*(1), 15495. https://doi.org/10.1038/s41598-017-15707-9

Hu, G.-M., Secario, M. K., & Chen, C.-M. (2019). SeQuery: an interactive graph database for visualizing the GPCR superfamily. *Database*, *2019*. https://doi.org/10.1093/database/baz073

Hu, G. M., Mai, T. L., & Chen, C. M. (2017). Visualizing the GPCR Network: Classification and Evolution. *Scientific Reports*, *7*(1), 1–15. https://doi.org/10.1038/s41598-017-15707-9

Huang, Y., Cai, J., Ji, L., & Li, Y. (2004). Classifying G-protein coupled receptors with bagging classification tree. *Computational Biology and Chemistry*, *28*(4), 275–280. https://doi.org/10.1016/j.compbiolchem.2004.08.001

Ikeda, M., Sugihara, M., & Suwa, M. (2018). SEVENS: a database for comprehensive GPCR genes obtained from genomes. *Biophysics and Physicobiology*, *15*(0), 104–110. https://doi.org/10.2142/biophysico.15.0_104

Inoue, Y., Ikeda, M., & Shimizu, T. (2004). Proteome-wide classification and identification of mammalian-type GPCRs by binary topology pattern. *Computational Biology and Chemistry*, *28*(1), 39–49. https://doi.org/10.1016/j.compbiolchem.2003.11.003

Iqbal, M. J., Faye, I., & Samir, B. B. (2016). Classification of GPCRs proteins using a statistical encoding method. *Proceedings of the International Joint Conference on Neural Networks*, *2016-Octob*, 1224–1228. https://doi.org/10.1109/IJCNN.2016.7727337

Isberg, V., Mordalski, S., Munk, C., Rataj, K., Harpsøe, K., Hauser, A. S., … Gloriam, D. E. (2016). GPCRdb: an information system for G protein-coupled receptors. *Nucleic Acids Research*, *44*(D1), D356-64. https://doi.org/10.1093/nar/gkv1178

Jo, M., & Jung, S. T. (2016). Engineering therapeutic antibodies targeting G-protein–coupled receptors. *Experimental & Molecular Medicine*, *48*(2), e207. https://doi.org/10.1038/emm.2015.105

Joachims, T. (n.d.). *A Service of zbw Making large-scale SVM learning practical*. Retrieved from http://hdl.handle.net/10419/77178www.econstor.eu

Karchin, R., Karplus, K., & Haussler, D. (2002). Classifying G-protein coupled receptors with support vector machines. *Bioinformatics (Oxford, England)*, *18*(1), 147–159. https://doi.org/10.1093/bioinformatics/18.1.147

Krishnan, A., Almén, M. S., Fredriksson, R., & Schiöth, H. B. (2012). The origin of GPCRs: identification of mammalian like Rhodopsin, Adhesion, Glutamate and Frizzled GPCRs in fungi. *PloS One*, *7*(1), e29817. https://doi.org/10.1371/journal.pone.0029817

Langenhan, T., Aust, G., & Hamann, J. (2013). Sticky signaling--adhesion class G protein-coupled receptors take the stage. *Science Signaling*, *6*(276), re3. https://doi.org/10.1126/scisignal.2003825

Langenhan, T., Barr, M. M., Bruchas, M. R., Ewer, J., Griffith, L. C., Maiellaro, I., … Monk, K. R. (2015). Model organisms in G protein-coupled receptor research. *Molecular Pharmacology*, *88*(3), 596–603. https://doi.org/10.1124/mol.115.098764

Li, Y. H., Xu, J. Y., Tao, L., Li, X. F., Li, S., Zeng, X., … Chen, Y. Z. (2016). SVM-Prot 2016: A Web-Server for Machine Learning Prediction of Protein Functional Families from Sequence Irrespective of Similarity. *PLOS ONE*, *11*(8), e0155290. https://doi.org/10.1371/journal.pone.0155290

Li, Z., Zhou, X., Dai, Z., & Zou, X. (2010). Classification of G-protein coupled receptors based

on support vector machine with maximum relevance minimum redundancy and genetic algorithm. *BMC Bioinformatics*, *11*, 325. https://doi.org/10.1186/1471-2105-11-325

Liao, Z., Ju, Y., & Zou, Q. (2016). Prediction of G Protein-Coupled Receptors with SVM-Prot Features and Random Forest. *Scientifica*, *2016*, 1–10. https://doi.org/10.1155/2016/8309253

Liu, Y., An, S., Ward, R., Yang, Y., Guo, X.-X., Li, W., & Xu, T.-R. (2016). G protein-coupled receptors as promising cancer targets. *Cancer Letters*, *376*(2), 226–239. https://doi.org/10.1016/j.canlet.2016.03.031

Munk, C., Isberg, V., Mordalski, S., Harpsøe, K., Rataj, K., Hauser, A. S., … Gloriam, D. E. (2016). GPCRdb: the G protein-coupled receptor database – an introduction. *British Journal of Pharmacology*, *16*, 2195–2207. https://doi.org/10.1111/bph.13509

Munoz, S., Guerrero, F. D., Kellogg, A., Heekin, A. M., & Leung, M. Y. (2017). Bioinformatic prediction of G protein-coupled receptor encoding sequences from the transcriptome of the foreleg, including the Haller's organ, of the cattle tick, Rhipicephalus australis. *PLoS ONE*, *12*(2), 1–22. https://doi.org/10.1371/journal.pone.0172326

Naveed, M., & Khan, A. U. (2012). GPCR-MPredictor: Multi-level prediction of G protein-coupled receptors using genetic ensemble. *Amino Acids*, *42*(5), 1809–1823. https://doi.org/10.1007/s00726-011-0902-6

Nordström, K. J. V., Sällman Almén, M., Edstam, M. M., Fredriksson, R., & Schiöth, H. B. (2011). Independent HHsearch, Needleman-Wunsch-based, and motif analyses reveal the overall hierarchy for most of the g protein-coupled receptor families. *Molecular Biology and Evolution*, *28*(9), 2471–2480. https://doi.org/10.1093/molbev/msr061

Palczewski, K., Kumasaka, T., Hori, T., Behnke, C. A., Motoshima, H., Fox, B. A., … Miyano, M. (2000). Crystal Structure of Rhodopsin: A G Protein-Coupled Receptor. *Science*, *289*(5480), 739–745. https://doi.org/10.1126/science.289.5480.739

Pándy-Szekeres, G., Munk, C., Tsonkov, T. M., Mordalski, S., Harpsøe, K., Hauser, A. S., … Gloriam, D. E. (2018). GPCRdb in 2018: Adding GPCR structure models and ligands. *Nucleic Acids Research*, *46*(D1), D440–D446. https://doi.org/10.1093/nar/gkx1109

Pearson, W. R., & Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, *85*(8), 2444–2448. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/3162770

Peng, Z.-L., Yang, J.-Y., & Chen, X. (2010). An improved classification of G-protein-coupled receptors using sequence-derived features. *BMC Bioinformatics*, *11*(1), 420. https://doi.org/10.1186/1471-2105-11-420

Poyner, D. R., & Hay, D. L. (2012). Secretin family (Class B) G protein-coupled receptors - from molecular to clinical perspectives. *British Journal of Pharmacology*, *166*(1), 1–3. https://doi.org/10.1111/j.1476-5381.2011.01810.x

Pruitt, K. D., Tatusova, T., & Maglott, D. R. (2007). NCBI reference sequences (RefSeq): A curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, *35*(SUPPL. 1). https://doi.org/10.1093/nar/gkl842

Qian, B., Soyer, O. S., Neubig, R. R., & Goldstein, R. A. (2003). Depicting a protein's two faces: GPCR classification by phylogenetic tree-based HMMs. *FEBS Letters*, *554*(1–2), 95–99. https://doi.org/10.1016/S0014-5793(03)01112-8

Raisley, B., Zhang, M., Hereld, D., & Hadwiger, J. A. (2004). A cAMP receptor-like G protein-coupled receptor with roles in growth regulation and development. *Developmental Biology*, *265*(2), 433–445. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/14732403

Sahin, M. E., Can, T., & Son, C. D. (2014). GPCRsort-responding to the next generation

sequencing data challenge: prediction of G protein-coupled receptor classes using only structural region lengths. *Omics: A Journal of Integrative Biology*, *18*(10), 636–644. https://doi.org/10.1089/omi.2014.0073

Schiöth, H. B., & Fredriksson, R. (2005). The GRAFS classification system of G-protein coupled receptors in comparative perspective. *General and Comparative Endocrinology*, *142*(1-2 SPEC. ISS.), 94–101. https://doi.org/10.1016/j.ygcen.2004.12.018

Sensoy, O., & Weinstein, H. (2015). A mechanistic role of Helix 8 in GPCRs: Computational modeling of the dopamine D2 receptor interaction with the GIPC1-PDZ-domain. *Biochimica et Biophysica Acta - Biomembranes*, *1848*(4), 976–983. https://doi.org/10.1016/j.bbamem.2014.12.002

Sonnhammer, E. L. L., Von Heijne, G., & Krogh, A. (1998). *A hidden Markov model for predicting transmembrane helices in protein sequences*. Retrieved from www.aaai.org

Theodoropoulou, M. C., Bagos, P. G., Spyropoulos, I. C., & Hamodrakas, S. J. (2008). gpDB: A database of GPCRs, G-proteins, effectors and their interactions. *Bioinformatics*, *24*(12), 1471–1472. https://doi.org/10.1093/bioinformatics/btn206

UniProt Consortium, T. U. (2008). The universal protein resource (UniProt). *Nucleic Acids Research*, *36*(Database issue), D190-5. https://doi.org/10.1093/nar/gkm895

Vassilatis, D. K., Hohmann, J. G., Zeng, H., Li, F., Ranchalis, J. E., Mortrud, M. T., … Gaitanaris, G. A. (2003). The G protein-coupled receptor repertoires of human and mouse. *Proceedings of the National Academy of Sciences of the United States of America*, *100*(8), 4903–4908. https://doi.org/10.1073/pnas.0230374100

Venkatakrishnan, A. J., Deupi, X., Lebon, G., Heydenreich, F. M., Flock, T., Miljus, T., … Babu, M. M. (2016). Diverse activation pathways in class A GPCRs converge near the G-protein-coupling region. *Nature*, *536*(7617), 484–487. https://doi.org/10.1038/nature19107

Vroling, B., Sanders, M., Baakman, C., Borrmann, A., Verhoeven, S., Klomp, J., … Vriend, G. (2011). GPCRDB: information system for G protein-coupled receptors. *Nucleic Acids Research*, *39*(Database issue), D309-19. https://doi.org/10.1093/nar/gkq1009

Wang, J., Gareri, C., & Rockman, H. A. (2018). G-protein-coupled receptors in heart disease. *Circulation Research*. Lippincott Williams and Wilkins. https://doi.org/10.1161/CIRCRESAHA.118.311403

Wolfram, S. (1984). Cellular automata as models of complexity. *Nature*, *311*(5985), 419–424. https://doi.org/10.1038/311419a0

Wu, H., Wang, C., Gregory, K. J., Han, G. W., Cho, H. P., Xia, Y., … Stevens, R. C. (2014). Structure of a class C GPCR metabotropic glutamate receptor 1 bound to an allosteric modulator. *Science (New York, N.Y.)*, *344*(6179), 58–64. https://doi.org/10.1126/science.1249489

Xiao, X., Wang, P., & Chou, K.-C. (2009). GPCR-CA: A cellular automaton image approach for predicting G-protein-coupled receptor functional classes. *Journal of Computational Chemistry*, *30*(9), 1414–1423. https://doi.org/10.1002/jcc.21163

Xue, C., Hsueh, Y. P., & Heitman, J. (2008, November). Magnificent seven: Roles of G protein-coupled receptors in extracellular sensing in fungi. *FEMS Microbiology Reviews*. https://doi.org/10.1111/j.1574-6976.2008.00131.x

Zamanian, M., Kimber, M. J., McVeigh, P., Carlson, S. A., Maule, A. G., & Day, T. A. (2011). The repertoire of G protein-coupled receptors in the human parasite Schistosoma mansoni and the model organism Schmidtea mediterranea. *BMC Genomics*, *12*, 596. https://doi.org/10.1186/1471-2164-12-596

Zhang, X. C., Liu, J., & Jiang, D. (2014). Why is dimerization essential for class-C GPCR function? New insights from mGluR1 crystal structure analysis. *Protein and Cell*, *5*(7), 492–495. https://doi.org/10.1007/s13238-014-0062-z

Appendices

APPENDIX A – PARSING DATA FROM GPCRDB

```
# To get the name of the children/lower level names for each family
import json
from pprint import pprint

with open('gpcrdb_006_001_001_children.json') as f:
      data = json.load(f)

#pprint(data)

fo = open('parse_006_001_001_children.csv','a')

#for i in data:

fo.write(data["slug"])
fo.write(",")
fo.write(data["name"])
fo.write("\n")

      fo.close()


#Collecting entry name, id, family name, species name and protein
sequence
import json
from pprint import pprint

with open('gpcrdb.org7.json') as f:
      data = json.load(f)

#pprint(data)

fo = open('gpcrdb_007.csv','a')

for i in data:

      fo.write(i["entry_name"])
      fo.write(",")
      fo.write(i["accession"])
      fo.write(",")
      fo.write(i["family"])
      fo.write(",")
      fo.write(i["species"])
      fo.write(",")
      fo.write(i["sequence"])
      fo.write("\n")
#adding 2nd column from a list to the other one when ID matches
# for comparing GPCRdb with GPCR-PEnDB
```

```
import csv

infile_list = open("classA_list.csv","r")

infile_id = open("classA_names.csv","r")

lines_list = infile_list.readlines()

lines_id = infile_id.readlines()

#print lines_list

lines_list = [item.split(",") for item in lines_list]
#lines_list = [item.split("\r\n") for item in lines_list]
#print lines_list

lines_id = [item.split("\r\n") for item in lines_id]
#print lines_id

fo = open("classA_4thlevel.csv","a")
#print lines_list[1]


for id_1 in lines_id:

      counter = 0
      #fo.write(id_1[0])
      #fo.write(",")

      for id_2 in lines_list:

            if id_1[0] == id_2[0]:

                  print id_1[0],id_2[1]

                  s = str(id_1[0]) + ',' + str(id_2[1])
                  fo.write(s)

            else:
                  counter = counter + 1
      #print counter
      if counter >299:
            print id_1[0]
            fo.write(id_1[0])
            fo.write('\n')

fo.close()
```

Appendix B – Non-GPCR IDs, Protein name in Python

```
import sys,subprocess , numpy
def nongpcr_id():
      infile = open('Non.fasta', 'r')
      lines = infile.readlines()
      seqs = []
      tempseq = ''
      for line in lines:
            if line[0] == '>' and not tempseq:
                  tempheader = line.strip()
            elif line[0] == '>':
                  seqs.append([tempheader,tempseq])
                  tempheader = line.strip()
                  tempseq = ''
            else:
                  tempseq = tempseq + line.strip()
      seqs.append([tempheader,tempseq])

# create a file for the IDs
      fo = open('non_cdhit_ID.csv','a')
#For each sequence
      for seq in seqs:
      #Write to temp file
            tempfile = open('temp.fasta', 'w')
            tempfile.write('%s\n%s'%(seq[0],seq[1]))
            tempfile.close()

            with open("temp.fasta") as gpcr:

                  for line in gpcr:
                        if line.startswith(">"):
                              line = line.replace(' ','')
                              header = line.split()
                              number = header[0].split('|')

                              print "Id:",number[1]

                              continue

                  fo.write(number[1])
                  fo.write('\n')

      fo.close()

nongpcr_id()


import sys,subprocess , numpy
def protein_name():
      infile = open('Non.fasta', 'r')
      lines = infile.readlines()
```

```
      seqs = []
      tempseq = ''
      for line in lines:
            if line[0] == '>' and not tempseq:
                  tempheader = line.strip()
            elif line[0] == '>':
                  seqs.append([tempheader,tempseq])
                  tempheader = line.strip()
                  tempseq = ''
            else:
                  tempseq = tempseq + line.strip()
      seqs.append([tempheader,tempseq])

# create a file for protein names
      fo = open('non_cdhit_protein_name.csv','a')
#For each sequence
      for seq in seqs:
      #Write to temp file
            tempfile = open('temp.fasta', 'w')
            tempfile.write('%s\n%s'%(seq[0],seq[1]))
            tempfile.close()


            with open("temp.fasta") as gpcr:

                  for line in gpcr:
                        if line.startswith(">"):
                              header = line.split('|')
                              number = header[2].split(' ')

                              print number[1]

                              continue

                  fo.write(number[1])
                  fo.write('\n')

      fo.close()

protein_name()

import sys,subprocess , numpy
def sequence():
      infile = open('Non.fasta', 'r')
      lines = infile.readlines()
      seqs = []
      tempseq = ''
      for line in lines:
            if line[0] == '>' and not tempseq:
                  tempheader = line.strip()
            elif line[0] == '>':
                  seqs.append([tempheader,tempseq])
```

```
                tempheader = line.strip()
                tempseq = ''
            else:
                tempseq = tempseq + line.strip()
        seqs.append([tempheader,tempseq])

# Create a file for the sequences
        fo = open('non_cdhit_sequences.csv','a')
#For each sequence
        for seq in seqs:
        #Write to temp file
            tempfile = open('temp.fasta', 'w')
            tempfile.write('%s\n%s'%(seq[0],seq[1]))
            tempfile.close()


            with open("temp.fasta") as gpcr:

                for line in gpcr:
                    if line.startswith(">"):

                        line = line.replace(' ','')
                        header = line.split()
                        number = header[0].split('|')


                        continue

                    else:
                        seq_length = len(line)
                        fo.write(line)
                        fo.write('\n')
                        print line


        fo.close()

sequence()
```

Appendix C – Taxonomy, IDs for Organisms in Python

```
import sys,subprocess , numpy

def strain():

        infile = open('SwissProtGPCRs.fasta', 'r')
        lines = infile.readlines()
        seqs = []
        tempseq = ''
        for line in lines:
            if line[0] == '>' and not tempseq:
                tempheader = line.strip()
```

```
            elif line[0] == '>':
                  seqs.append([tempheader,tempseq])
                  tempheader = line.strip()
                  tempseq = ''
            else:
                  tempseq = tempseq + line.strip()
      seqs.append([tempheader,tempseq])


# create a file for strain
      fo1 = open('11_13_gpcr_organism_name.csv','a')
      fo = open('11_13_gpcr_virus_name.csv','a')
#For each sequence
      for seq in seqs:
      #Write to temp file
            tempfile = open('temp.fasta', 'w')
            tempfile.write('%s\n%s'%(seq[0],seq[1]))
            tempfile.close()


            with open("temp.fasta") as gpcr:

                  xlist = []
                  for line in gpcr:
                        if line.startswith(">"):
                        #     print line
                              header = line.split('=')
                              #organism = header[1].split('GN')[0]
                              organism = header[1][0:-3]
                              column1 = header[0].split('|')
                              column1 = column1[1]
                              altprint = 0
                              sero = ''
                              strain = ''
                              if 'virus' in organism or 'Virus' in
organism:
                                    altprint = 1
                              elif '(' in organism:
                                    ols = organism.split('(')
                                    gss = ols[0].strip().split()
                                    if len(gss) == 2:
                                          genus = gss[0]
                                          species = gss[1]
                                    elif len(gss) > 2:
                                          genus = gss[0]
                                          species = gss[1]
                                          sero = ' '.join(gss[2:])
                                    else:
                                          altprint = 1

                                    strain = ols[1].split(')')[0]
                              else:
```

```
                                    gss = organism.strip().split()
                                    if len(gss) == 2:
                                          genus = gss[0]
                                          species = gss[1]
                                    elif len(gss) > 2:
                                          genus = gss[0]
                                          species = gss[1]
                                          sero = ' '.join(gss[2:])
                                    else:
                                           altprint = 1
                              if altprint == 1:
                                    fo.write(','.join([column1,organism]))
                                    fo.write('\n')
                              else:

      fo1.write(','.join([column1,genus,species,sero,strain]))
                                    fo1.write('\n')

      fo.close()
      fo1.close()
strain()
```

APPENDIX D: PERCENTAGES OF AMINO ACIDS IN PYTHON

```
import sys,subprocess , numpy
def count_aa():
      infile = open('non_five.fasta', 'r')
      lines = infile.readlines()
      seqs = []
      tempseq = ''
      for line in lines:
            if line[0] == '>' and not tempseq:
                  tempheader = line.strip()
            elif line[0] == '>':
                  seqs.append([tempheader,tempseq])
                  tempheader = line.strip()
                  tempseq = ''
            else:
                  tempseq = tempseq + line.strip()
      seqs.append([tempheader,tempseq])

# create a file with tm = 6 or 8
      tmhmm_file = open('tmhmm.fasta','w')
      fo = open('swiss_id.csv','a')
#For each sequence
      for seq in seqs:
      #Write to temp file
            tempfile = open('temp.fasta', 'w')
            tempfile.write('%s\n%s'%(seq[0],seq[1]))
            tempfile.close()

      #Write to temp file call tmhmm
```

```
          comm = '/export/home/kbegum/gpcr_code/tmhmm-2.0c/bin/tmhmm
temp.fasta -noplot'
          process = subprocess.Popen(comm, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
          process.wait()
          lin = process.stdout.readlines()

          with open("temp.fasta") as gpcr:

                counts = {}
                keys = ["AA", "C", "D", "E", "F", "G", "H", "I", "K",
"L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y"]
                keys1 = ["X"]
                for char in keys:
                     counts[char] = 0

                for line in gpcr:
                     if line.startswith(">"):
                          line = line.replace(' ','')
                          header = line.split()
                          number = header[0].split('|')

                     else:
                          seq_length = len(line)
                          z = str(seq_length)
                          print seq_length
                     for char in line.strip():


                          if char in keys:
                               counts[char] += 1
                          total = float(sum(counts.values()))
                          others = seq_length - total
                          s_seq_length = str(others)


                toReturn = ''


                for key in keys:
                     aa_per = (counts[key])
                     toReturn = toReturn + '%.4f'%aa_per + ','

                fo.write(''.join(str(x) for x in toReturn))
                #fo.write(s_seq_length)
                fo.write('\n')
                return toReturn
     fo.close()

count_aa()
```

Appendix E – Dipeptide count of amino acids in Python

Dipeptide count code:

```
import re
complete=[]
count=0
seq=""
for line in open("Non.fasta"):
    if line.startswith(">"):
        if count>0:
            complete.append(seq)
        header = line
        header = header.replace("\n","")
        header = header.replace("\r","")
        complete.append(header)
        seq=""
        count=count+1
    else:
        hold = line
        hold = hold.replace("\n","")
        hold = hold.replace("\r","")
        seq = seq + hold

complete.append(seq)

nucleotides = ["A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M",
"N", "P", "Q", "R", "S", "T", "V", "W", "Y"]
'''
words1 = []
for x in nucleotides:
        word=x
        words1.append(word)

for e in complete:
    for n in words1:
        if e.startswith(">"):
            header = e
        else:
            num=e.count(n)
            file = open("1_words.csv","a")
            header = header.replace(",",";")
            l=len(e)
            freq = (num/float(l))
            file.write(header + "," + str(l) + "," + n + "," +
str(num) + "," + str(freq) + "\n")
            file.close()
'''
file = open("2_mist1.csb","a")
words2 = []
p = []
```

```
for x in nucleotides:
    for y in nucleotides:
        word=x+y
#     word = word + ','
        words2.append(word)

#print str.replace("'","")

for e in complete:
    length = len(e)
    for n in words2:
        if e.startswith(">"):
            header = e
        else:
            num = 0
            idx = 0
            while True:
                idx = e.find(n, idx)
                if idx >= 0:
                    num += 1
                    idx += 1
                else:
                    break
            p.append(num)

     chunks = [p[x:x+400] for x in range(0, len(p), 400)]

c = 0
for i in chunks:
     file.write(str(chunks[c]))
     file.write('\n')

     c = c + 1
file.close()
```

Organism table ID adjustments:

```
import re
infile1 = open('11_29_id_organism.csv','r')
infile2 = open('11_29_organism_table.csv','r')
lines1 = infile1.read().splitlines()
lines2 = infile2.readlines()

fo = open('1130_organism.csv','a')
d = ''

#m = [m.replace('\r\n','') for m in lines1]


for line in lines1:
     m = line.split(',')
     for lines in lines2:
          c = lines.split(',')
          if m[1:5] == c[1:5]:
               d = str(m[0]) + ',' + str(c[0])
               print d
               fo.write(d)
               fo.write('\n')
fo.close()
```

Appendix F – Length using TMHMM2.0 in Python

```
import sys,subprocess , numpy
def length_aa():
     infile = open('non_five.fasta', 'r')
     lines = infile.readlines()
     seqs = []
     tempseq = ''
     for line in lines:
          if line[0] == '>' and not tempseq:
               tempheader = line.strip()
          elif line[0] == '>':
               seqs.append([tempheader,tempseq])
               tempheader = line.strip()
               tempseq = ''
          else:
               tempseq = tempseq + line.strip()
     seqs.append([tempheader,tempseq])

# create a file with tm = 6 or 8
     tmhmm_file = open('tmhmm.fasta','w')
     #fo = open('aa_count_mist_swiss.csv','a')
#For each sequence
     for seq in seqs:
     #Write to temp file
          tempfile = open('temp.fasta', 'w')
          tempfile.write('%s\n%s'%(seq[0],seq[1]))
          tempfile.close()
```

```
     #Write to temp file call tmhmm


          comm = '/export/home/kbegum/gpcr_code/tmhmm-2.0c/bin/tmhmm
temp.fasta -noplot'
          process = subprocess.Popen(comm, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
          process.wait()
          lin = process.stdout.readlines()

          with open("temp.fasta") as gpcr:

               counts = {}
               keys = ["A", "C", "D", "E", "F", "G", "H", "I", "K",
"L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y"]
               keys1 = ["X"]
               for char in keys:
                    counts[char] = 0

               for line in gpcr:
                    if line.startswith(">"):
                         line = line.replace(' ','')
                         header = line.split()
                         number = header[0].split('|')

               #          print number[2]

                         continue
               #          fo.write(number[1])
               #          fo.write('\n')
                    else:
                         seq_length = len(line)
                         z = str(seq_length)
          #              fo.write(z)
          #              fo.write('\n')
                         #print seq_length
                    for char in line.strip():


                         if char in keys:
                              counts[char] += 1
                         total = float(sum(counts.values()))
                         others = seq_length - total
                         s_seq_length = str(others)


               toReturn = ''


               for key in keys:
                    aa_per = (counts[key])
                    toReturn = toReturn + str(aa_per) + ','
```

```
                #fo.write('ID')
                #fo.write('\t')
                #fo.write(number[2])

                #print toReturn
                #fo.write('\n')
      #         fo.write(''.join(str(x) for x in toReturn))
                #fo.write(s_seq_length)
      #         fo.write('\n')

                print toReturn
#               return toReturn
      fo.close()

length_aa()
```

Appendix G – MySQL commands for creating tables and populating data

PROTEINS table:

```
LOAD DATA LOCAL INFILE
'/export/home/kbegum/gpcr_code/merge_gpcr_final_1.csv' INTO TABLE
GPCR2 FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (PROTEIN_ID,
Name,Organism_ID,IUPHAR_ID,GRAFS_ID,Sequence,Length,A,C,D,E,F,G,H,I,K,
L,M,N,P,Q,R,S,T,V,W,Y,Other,PROTEIN_TYPE);
```

Dipeptide table:

```
LOAD DATA LOCAL INFILE
'/export/home/kbegum/gpcr_code/merge_gpcr_final_1.csv' INTO TABLE
GPCR2 FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (GPCR_ID,
Name,Organism_ID,IUPHAR_ID,GRAFS_ID,Sequence,Length,A,C,D,E,F,G,H,I,K,
L,M,N,P,Q,R,S,T,V,W,Y,Other,PROTEIN_TYPE);
CREATE TABLE AA_Dipeptide( PROTEIN_ID varchar(25) PRIMARY KEY, A1
varchar(25), C1 varchar(25), D1 varchar(25), E1 varchar(25), F1
varchar(25), G1 varchar(25), H1 varchar(25), I1 varchar(25), K1
varchar(25), L1 varchar(25), M1 varchar(25), N1 varchar(25), P1
varchar(25), Q1 varchar(25), R1 varchar(25), S1 varchar(25), T1
varchar(25), V1 varchar(25), W1 varchar(25), Y1 varchar(25), Others1
varchar(25), AA varchar(25),  AC varchar(25),  AD varchar(25),  AE
varchar(25),  AF varchar(25),  AG varchar(25),  AH varchar(25),  AI
varchar(25),  AK varchar(25),AL varchar(25),AM varchar(25), AN
varchar(25),  AP varchar(25),  AQ varchar(25),  AR varchar(25),  AS1
varchar(25),  AT varchar(25),  AV varchar(25),  AW varchar(25),  AY
varchar(25),  CA varchar(25),  CC varchar(25),  CD varchar(25),  CE
varchar(25),  CF varchar(25),  CG varchar(25),  CH varchar(25),  CI
varchar(25),  CK varchar(25),  CL varchar(25),  CM varchar(25),  CN
varchar(25),  CP varchar(25),  CQ varchar(25),  CR varchar(25),  CS
varchar(25),  CT varchar(25),  CV varchar(25),  CW varchar(25),  CY
```

```
varchar(25),  DA varchar(25),  DC varchar(25),  DD varchar(25),  DE
varchar(25),  DF varchar(25),  DG varchar(25),  DH varchar(25),  DI
varchar(25),  DK varchar(25),  DL varchar(25),  DM varchar(25),  DN
varchar(25),  DP varchar(25),  DQ varchar(25),  DR varchar(25),  DS
varchar(25),  DT varchar(25),  DV varchar(25),  DW varchar(25),  DY
varchar(25),  EA varchar(25),  EC varchar(25),  ED varchar(25),  EE
varchar(25),  EF varchar(25),  EG varchar(25),  EH varchar(25),  EI
varchar(25),  EK varchar(25),  EL varchar(25),  EM varchar(25),  EN
varchar(25),  EP varchar(25),  EQ varchar(25),  ER varchar(25),  ES
varchar(25),  ET varchar(25),  EV varchar(25),  EW varchar(25),  EY
varchar(25),  FA varchar(25),  FC varchar(25),  FD varchar(25),  FE
varchar(25),  FF varchar(25),  FG varchar(25),  FH varchar(25),  FI
varchar(25),  FK varchar(25),  FL varchar(25),  FM varchar(25),  FN
varchar(25),  FP varchar(25),  FQ varchar(25),  FR varchar(25),  FS
varchar(25),  FT varchar(25),  FV varchar(25),  FW varchar(25),  FY
varchar(25),  GA varchar(25),  GC varchar(25),  GD varchar(25),  GE
varchar(25),  GF varchar(25),  GG varchar(25),  GH varchar(25),  GI
varchar(25),  GK varchar(25),  GL varchar(25),  GM varchar(25),  GN
varchar(25),  GP varchar(25),  GQ varchar(25),  GR varchar(25),  GS
varchar(25),  GT varchar(25),  GV varchar(25),  GW varchar(25),  GY
varchar(25),  HA varchar(25),  HC varchar(25),  HD varchar(25),  HE
varchar(25),  HF varchar(25),  HG varchar(25),  HH varchar(25),  HI
varchar(25),  HK varchar(25),  HL varchar(25),  HM varchar(25),  HN
varchar(25),  HP varchar(25),  HQ varchar(25),  HR varchar(25),  HS
varchar(25),  HT varchar(25),  HV varchar(25),  HW varchar(25),  HY
varchar(25),  IA varchar(25),  IC varchar(25),  ID varchar(25),  IE
varchar(25),  IF1 varchar(25),  IG varchar(25),  IH varchar(25),  II
varchar(25),  IK varchar(25),  IL varchar(25),  IM varchar(25),  IN1
varchar(25),  IP varchar(25),  IQ varchar(25),  IR varchar(25),  IS1
varchar(25),  IT varchar(25),  IV varchar(25),  IW varchar(25),  IY
varchar(25),  KA varchar(25),  KC varchar(25),  KD varchar(25),  KE
varchar(25),  KF varchar(25),  KG varchar(25),  KH varchar(25),  KI
varchar(25),  KK varchar(25),  KL varchar(25),  KM varchar(25),  KN
varchar(25),  KP varchar(25),  KQ varchar(25),  KR varchar(25),  KS
varchar(25),  KT varchar(25),  KV varchar(25),  KW varchar(25),  KY
varchar(25),  LA varchar(25),  LC varchar(25),  LD varchar(25),  LE
varchar(25),  LF varchar(25),  LG varchar(25),  LH varchar(25),  LI
varchar(25),  LK varchar(25),  LL varchar(25),  LM varchar(25),  LN
varchar(25),  LP varchar(25),  LQ varchar(25),  LR varchar(25),  LS
varchar(25),  LT varchar(25),  LV varchar(25),  LW varchar(25),  LY
varchar(25),  MA varchar(25),  MC varchar(25),  MD varchar(25),  ME
varchar(25),  MF varchar(25),  MG varchar(25),  MH varchar(25),  MI
varchar(25),  MK varchar(25),  ML varchar(25),  MM varchar(25),  MN
varchar(25),  MP varchar(25),  MQ varchar(25),  MR varchar(25),  MS
varchar(25),  MT varchar(25),  MV varchar(25),  MW varchar(25),  MY
varchar(25),  NA varchar(25),  NC varchar(25),  ND varchar(25),  NE
varchar(25),  NF varchar(25),  NG varchar(25),  NH varchar(25),  NI
varchar(25),  NK varchar(25),  NL varchar(25),  NM varchar(25), NN
varchar(25),  NP varchar(25),  NQ varchar(25),  NR varchar(25),  NS
varchar(25),  NT varchar(25),  NV varchar(25),  NW varchar(25),  NY
varchar(25),  PA varchar(25),  PC varchar(25),  PD varchar(25),  PE
varchar(25),  PF varchar(25),  PG varchar(25),  PH varchar(25),  PI
```

```
varchar(25),  PK varchar(25),  PL varchar(25),  PM varchar(25),  PN
varchar(25),  PP varchar(25),  PQ varchar(25),  PR varchar(25),  PS
varchar(25),  PT varchar(25),  PV varchar(25),  PW varchar(25),  PY
varchar(25),  QA varchar(25),  QC varchar(25),  QD varchar(25),  QE
varchar(25),  QF varchar(25),  QG varchar(25),  QH varchar(25),  QI
varchar(25),  QK varchar(25), QL varchar(25),  QM varchar(25),  QN
varchar(25),  QP varchar(25),  QQ varchar(25),  QR varchar(25),  QS
varchar(25),  QT varchar(25),  QV varchar(25),  QW varchar(25),  QY
varchar(25),  RA varchar(25),  RC varchar(25),  RD varchar(25),  RE
varchar(25),  RF varchar(25),  RG varchar(25),  RH varchar(25),  RI
varchar(25),  RK varchar(25),  RL varchar(25),  RM varchar(25),  RN
varchar(25),  RP varchar(25),  RQ varchar(25),  RR varchar(25),  RS
varchar(25),  RT varchar(25),  RV varchar(25),  RW varchar(25),  RY
varchar(25),  SA varchar(25),  SC varchar(25),  SD varchar(25),  SE
varchar(25),  SF varchar(25),  SG varchar(25),  SH varchar(25),  SI
varchar(25),  SK varchar(25),  SL varchar(25),  SM varchar(25),  SN
varchar(25),  SP varchar(25),  SQ varchar(25),  SR varchar(25),  SS
varchar(25),  ST varchar(25),  SV varchar(25),  SW varchar(25),  SY
varchar(25),  TA varchar(25),  TC varchar(25),  TD varchar(25),  TE
varchar(25),  TF varchar(25),  TG varchar(25),  TH varchar(25),  TI
varchar(25),  TK varchar(25),  TL varchar(25),  TM varchar(25),  TN
varchar(25),  TP varchar(25),  TQ varchar(25),  TR varchar(25),  TS
varchar(25),  TT varchar(25),  TV varchar(25),  TW varchar(25),  TY
varchar(25),  VA varchar(25),  VC varchar(25),  VD varchar(25),  VE
varchar(25),  VF varchar(25),  VG varchar(25),  VH varchar(25),  VI
varchar(25),  VK varchar(25),  VL varchar(25),  VM varchar(25),  VN
varchar(25), VP varchar(25),  VQ varchar(25),  VR varchar(25),  VS
varchar(25),  VT varchar(25),  VV varchar(25),  VW varchar(25),  VY
varchar(25),  WA varchar(25),  WC varchar(25),  WD varchar(25),  WE
varchar(25),  WF varchar(25),  WG varchar(25),  WH varchar(25),  WI
varchar(25),  WK varchar(25),  WL varchar(25),  WM varchar(25),  WN
varchar(25),  WP varchar(25),  WQ varchar(25),  WR varchar(25),  WS
varchar(25),  WT varchar(25),  WV varchar(25),  WW varchar(25),  WY
varchar(25),  YA varchar(25),  YC varchar(25),  YD varchar(25),  YE
varchar(25),  YF varchar(25),  YG varchar(25),  YH varchar(25),  YI
varchar(25),  YK varchar(25),  YL varchar(25),  YM varchar(25),  YN
varchar(25),  YP varchar(25),  YQ varchar(25),  YR varchar(25),  YS
varchar(25),  YT varchar(25),  YV varchar(25),  YW varchar(25),  YY
varchar(25));


LOAD DATA LOCAL  INFILE '/export/home/kbegum/gpcr_code/dekhajak1.csv'
INTO TABLE AA_Dipeptide FIELDS TERMINATED BY ',' LINES TERMINATED BY
'\n'
(PROTEIN_ID,A1,C1,D1,E1,F1,G1,H1,I1,K1,L1,M1,N1,P1,Q1,R1,S1,T1,V1,W1,Y
1,Others1,AA, AC, AD, AE, AF, AG, AH, AI, AK, AL, AM, AN, AP, AQ, AR,
AS1, AT, AV, AW, AY, CA, CC, CD, CE, CF, CG, CH, CI, CK, CL, CM, CN,
CP, CQ, CR, CS, CT, CV, CW, CY, DA, DC, DD, DE, DF, DG, DH, DI, DK,
DL, DM, DN, DP, DQ, DR, DS, DT, DV, DW, DY, EA, EC, ED, EE, EF, EG,
EH, EI, EK, EL, EM, EN, EP, EQ, ER, ES, ET, EV, EW, EY, FA, FC, FD,
FE, FF, FG, FH, FI, FK, FL, FM, FN, FP, FQ, FR, FS, FT, FV, FW, FY,
GA, GC, GD, GE, GF, GG, GH, GI, GK, GL, GM, GN, GP, GQ, GR, GS, GT,
```

```
GV, GW, GY, HA, HC, HD, HE, HF, HG, HH, HI, HK, HL, HM, HN, HP, HQ,
HR, HS, HT, HV, HW, HY, IA, IC, ID, IE, IF1, IG, IH, II, IK, IL, IM,
IN1, IP, IQ, IR, IS1, IT, IV, IW, IY, KA, KC, KD, KE, KF, KG, KH, KI,
KK, KL, KM, KN, KP, KQ, KR, KS, KT, KV, KW, KY, LA, LC, LD, LE, LF,
LG, LH, LI, LK, LL, LM, LN, LP, LQ, LR, LS, LT, LV, LW, LY, MA, MC,
MD, ME, MF, MG, MH, MI, MK, ML, MM, MN, MP, MQ, MR, MS, MT, MV, MW,
MY, NA, NC, ND, NE, NF, NG, NH, NI, NK, NL, NM, NN, NP, NQ, NR, NS,
NT, NV, NW, NY, PA, PC, PD, PE, PF, PG, PH, PI, PK, PL, PM, PN, PP,
PQ, PR, PS, PT, PV, PW, PY, QA, QC, QD, QE, QF, QG, QH, QI, QK, QL,
QM, QN, QP, QQ, QR, QS, QT, QV, QW, QY, RA, RC, RD, RE, RF, RG, RH,
RI, RK, RL, RM, RN, RP, RQ, RR, RS, RT, RV, RW, RY, SA, SC, SD, SE,
SF, SG, SH, SI, SK, SL, SM, SN, SP, SQ, SR, SS, ST, SV, SW, SY, TA,
TC, TD, TE, TF, TG, TH, TI, TK, TL, TM, TN, TP, TQ, TR, TS, TT, TV,
TW, TY, VA, VC, VD, VE, VF, VG, VH, VI, VK, VL, VM, VN, VP, VQ, VR,
VS, VT, VV, VW, VY, WA, WC, WD, WE, WF, WG, WH, WI, WK, WL, WM, WN,
WP, WQ, WR, WS, WT, WV, WW, WY, YA, YC, YD, YE, YF, YG, YH, YI, YK,
YL, YM, YN, YP, YQ, YR, YS, YT, YV, YW, YY);

CREATE TABLE Organism_v2 (Organism_ID varchar(25) PRIMARY KEY, Genus
varchar(25) NULL, Species varchar(60) NULL, Serotype varchar(60) NULL,
Strain varchar(100) NULL, Common_name varchar(100) NULL, Frequency
varchar(25));
LOAD DATA LOCAL  INFILE
'/export/home/kbegum/gpcr_code/11_18_new_organismid.csv' INTO TABLE
Organism_v2 FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n'
(Organism_ID, Genus, Species, Serotype, Strain, Common_name,
Frequency);
```

Updating Organism table:

```
CREATE TABLE Organism_v2 (Organism_ID varchar(25) PRIMARY KEY, Genus
varchar(25) NULL, Species varchar(60) NULL, Serotype varchar(60) NULL,
Strain varchar(100) NULL, Common_name varchar(100) NULL, Frequency
varchar(25));
LOAD DATA LOCAL  INFILE
'/export/home/kbegum/gpcr_code/11_18_new_organismid.csv' INTO TABLE
Organism_v2 FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n'
(Organism_ID, Genus, Species, Serotype, Strain, Common_name,
Frequency);
```

Updating columns of Organism ID:

```
UPDATE PROTEINS SET Organism_ID = lpad(Organism_ID,5,0)
CREATE TABLE temp_table (PROTEIN_ID varchar(25) PRIMARY KEY,
Organism_ID varchar(25) NULL, Frequency varchar(25));
LOAD DATA LOCAL  INFILE
'/export/home/kbegum/gpcr_code/1130_organism_sql.csv' INTO TABLE
temp_table FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n'
(PROTEIN_ID,Organism_ID);

UPDATE PROTEINS
INNER JOIN temp_table on temp_table.PROTEIN_ID = PROTEINS.PROTEIN_ID
```

```
SET PROTEINS.Organism_ID = temp_table.Organism_ID;

DROP TEMPORARY TABLE your_temp_table;
```

TMHMM length table:

```
CREATE TABLE TMHMM_Length (PROTEIN_ID varchar(25) PRIMARY KEY, N_term
varchar(25),IL1 varchar(25),IL2 varchar(25),IL3 varchar(25),TM1
varchar(25),TM2 varchar(25),TM3 varchar(25),TM4 varchar(25),TM5
varchar(25),TM6 varchar(25),TM7 varchar(25),OL1 varchar(25),OL2
varchar(25),OL3 varchar(25),C_term varchar(25),Length varchar(25));
LOAD DATA LOCAL  INFILE
'/export/home/kbegum/gpcr_code/11_28_tmhmm_length.csv' INTO TABLE
TMHMM_Length FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n'
(PROTEIN_ID, N_term, IL1, IL2,
IL3,TM1,TM2,TM3,TM4,TM5,TM6,TM7,OL1,OL2,OL3,C_term,Length);
```

APPENDIX H: WEB-SERVER INPUT PAGE

```
$def with (setup)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

          <meta http-equiv="content-type" content="text/html;
charset=utf-8" />
          <meta name="description" content="Short page description
here." />
          <meta name="keywords" content="keyword1, keyword2,
keyword3" />
<head>
    <title>UTEP GPCR-PEnDB</title>
       $:{setup['css']}
       $:{setup['favIco']}

      <script  language = "javascript"  type = "text/javascript">

           if (window.XMLHttpRequest) {
           Request = new XMLHttpRequest();
           } else if (window.ActiveXObject) {
           Request = new ActiveXObject("Microsoft.XMLHTTP");
        }
      window.onbeforeunload = function(){ window.scrollTo(0,0); }

      var file = "{% static 'json/html-table-elements.json' %}"
         </script>

      <script>

         function hidediv(){
              var mydiv = document.getElementById("param");
              if (mydiv == null)
```

```
                {  alert("Sorry can't find your div");
                   return;
                }
                 mydiv.style.display="none";
          }
         function showdiv(){
              var mydiv = document.getElementById("param");
              if (mydiv == null)
                 {  alert("Sorry can't find your div");
                    return;
                 }
              mydiv.style.display="table";

          }
          </script>
      <hr>
</head>

<body onLoad = "initLoad()" id="loadBody">
   <div id="page-wrap">
      <div id="header-wrap">$:{setup['banner']}  </div>
         $:{setup['navBar']}
      <table border="1">
           <div id="spinner" class="spinner" style="display:none;">
           <img id="img-spinner" align="left" src="/static/img/ajax-
loader.gif" alt="Loading"/>...One Moment Please...
           </div>
      </table>

    <center>

<h2>GPCR-PEnDB</h2>


                <div id = "searchform">
                <table width="950"   border="0" cellpadding="0"
cellspacing="0">
                     <tr><td colspan="0" width="125" style="vertical-
align:baseline">
The database contains more than
3000 confirmed GPCR (CG) and 3500 non-GPCR (CN) from more than 1200
different
organisms including bacteria and viruses. About half of the non-GPCR
sequences are
transmembrane proteins (CNT). Each protein, with a unique
identification number, is linked
to its source organism, gene name, protein name, sequence length, and
other features such
as amino acid and dipeptide compositions. For the GPCRs, the lengths
of characteristic
structural regions (i.e., N-terminal, C-terminal, seven transmembrane
helices, and the
```

```
extracellular and intracellular loops) are also provided. If
available, GPCR family
classification information is also included.


                    </td></tr>


</table>


<center>
                <div id = "searchform">
                <table width="950"   border="0" cellpadding="0"
cellspacing="0">
                    <tr><td colspan="0" width="125" style="vertical-
align:baseline"><h3>Quick Search</h3>
<td>
<a href="./database_instructions" target = "./database">(Help)</a>
</td>

                    </td></tr>


</table>

                <table width="950" border="1" cellpadding="8px"
cellspacing="1" bordercolor ="green">
                <tr><td>

                <form action=""  method="POST" id="post-form"
enctype="multipart/form-data">
                <div style="margin:0;padding:0">
                     <input name="authenticity_token" type="hidden"
value="6656e1052cf32fa79dbc6f644484e1d4b3980d46" />
                </div>

                     <input  style="width: 300px;" id="search_term"
name="search_term" type="text" placeholder="e.g. Name, Organism,
Protein ID"/>
                <select id="qt" name="qt">
                     <option value='all'>Show All Entries</option>
                     <option value='amino_acid_composition'>Amino Acid
Percentage</option>
                     <option
value='classification'>Classification</option>
                     <option value='entry_name'>Entry Name</option>
                     <option value='protein_name'>Protein
Name</option>
                     <option value='gene_name'>Gene Name</option>
                     <option    value='seq_length'>Length</option>
```

```
                    <option value='organism'>Organisms</option>

                    <option    value='protein_id'>UniProtKB
ID</option>


                </select>
                <input type="submit" value="Quick Search"
id="submitSequence"></input>



                </td></tr>
                     </table>



                <table width="950" border="0" cellpadding="0"
cellspacing="0">

                <tr><td colspan="6"><h3>Advanced Search
                     <form><input type="button" value="show"
onclick="showdiv()"/>
                          <input type="button" value="hide"
onclick="hidediv()"/>
                                        </form></h3></td></tr>
                </table>



                <table cellspacing=1 width="950" class='bodytext'
border=1 bordercolor="green" id="param" style="display:none">

                <form action="" method="POST" id="post-form"
enctype="multipart/form-data">
                    <div style="margin:0;padding:0">
                    <input name="authenticity_token" type="hidden"
value="49ef981d37a1eeae4cd8efdbc9887472583f2883" /></div>


                    <tr>
                    <td class="yellowcell"
align="center"><b>Organism</b></td>
                    <td><input id="a_organism" name="a_organism"
size="30" type="text" placeholder ="e.g. Human OR Mouse ..." /></td>
                    </tr>



                    <tr>
```

```
                    <td class="yellowcell"
align="center"><b>Protein</b></td>
                    <td><input id="a_pname" name="a_pname" size="30"
type="text" placeholder ="e.g. Opsin ..." /></td>
                    </tr>
                    <tr>
                    <td class="yellowcell"
align="center"><b>Gene</b></td>
                    <td><input id="a_gname" name="a_gname" size="30"
type="text" placeholder ="e.g. grk ..." /></td>
                    </tr>



                    <tr>
                    <td class="yellowcell" align =
"center"><b>Sequence Length</b></td>
                    <td><nobr>
                         <select id="length_range"
name="length_range" onChange="changeSecond('length');"
onLoad="changeSecond('length');">
                              <option value='equal_to'>Equal
to</option>
                              <option value='greater_than'>Greater
than</option>
                              <option value='less_than'>Less
than</option>
                         </select>
                         <input id="a_length" name="a_length"
size="10" type="text" />
                         <div style='display:inline;
visibility:hidden;' id=length_div > and <input disabled="disabled"
id="length_second" name="length[second]" size="10" type="text" />
                         </div>
                    </nobr></td>
                    </tr>



                    <tr>
                    <td class="yellowcell" align = "center"
><b>Category</b></td>


               <td><class = "yellowcell"/><b>GPCRs</b><br>


               <input type="checkbox" name="pro_type" id = "cg_type"
value = "cg_type" /> Full length 
```

```
                    <input type="checkbox" name="pro_type" id =
"cgf_type" value = "cgf_type" /> Fragments   <br>


                <class = "yellowcell"/><b> Non-GPCRs</b><br>

                    <input type="checkbox" name="pro_type"
id="cnt_type" value="cnt_type"  />Transmembrane 


                     <input type="checkbox" name="pro_type"
id="cn_type" value="cn_type"  />Non-transmembrane</td>


    </tr>



                    <tr>
                    <td class="yellowcell" align = "center"><b>GRAFS
family</b></td>

                    <td>


                        <input id="rho" type="checkbox" name="grafs"
value="Rhodopsin" /> Rhodopsin 
                        <input id="adh" type="checkbox" name="grafs"
value="Adhesion" />Adhesion 
                        <input id="sec" type="checkbox" name="grafs"
value="Secretin" /> Secretin 
                        <input id="glu" type="checkbox" name="grafs"
value="Glutamate"  />  Glutamate 
                        <input id = "fung" type ="checkbox" name =
"grafs" value ="fungal"/> Fungal pheromone 
                        <input id = "camp" type="checkbox" name =
"grafs" value = "camp"/> cAMP receptor 
                        <input id="fri" type="checkbox" name="grafs"
value="Frizzled" /> Frizzled <br>
                        <input id="taste2r" type="checkbox"
name="grafs" value="taste2r" /> Taste2R 
                    </td>
                    </tr>



                    <tr>
                    <td class="yellowcell" align = "center"><b>IUPHAR
family</b></td>
```

```
                    <td><input id="clA"  type="checkbox"
name="iuphar"  value="ClassA"  />  Class A  
                        <input id="clB" type="checkbox" name="iuphar"
value="ClassB" />Class B  
                        <input id="clC" type="checkbox" name="iuphar"
value="ClassC" /> Class C  
                        <input id="clD" type="checkbox" name="iuphar"
value="ClassD" />Class D  
                        <input id="clE" type="checkbox" name="iuphar"
value="ClassE" />Class E  
                        <input id="clF" type="checkbox" name="iuphar"
value="ClassF" />Class F  
                        <input id = "clt2R" type="checkbox" name =
"iuphar" value="ClassT2R" /> Class T2R  

                    </td>
                    </tr>
                    <tr>
                    <td class="yellowcell" align="center"><b>Sub-
family</b></td>
                    <td><input id="a_sub_fam" name="a_sub_fam"
size="30" type="text" placeholder ="e.g. Peptide ..." /></td>
                    </tr>
                    <tr>
                    <td class="yellowcell" align="center"><b>Sub-sub-
family</b></td>
                    <td><input id="a_sub_sub_fam"
name="a_sub_sub_fam" size="30" type="text" placeholder ="e.g.
Calcitonin ..." /></td>
                    </tr>
                    <tr>
                    <td class="yellowcell" align="center"><b>Sub-
type</b></td>
                    <td><input id="a_sub_type" name="a_sub_type"
size="30" type="text" placeholder ="e.g. CRF ..." /></td>
                    </tr>



                    <tr>
                    <td class="yellowcell" align = "center" ><b>Amino
acid percentage</b></td>

                    <td><input id="aa" name="aa" size="30"
type="text" placeholder ="e.g. A>2 ..." /></td>
                    </tr>

                    <tr>
                    <td class="yellowcell" align =
"center"><b>Dipeptide percentage</b></td>
```

```
                        <td><input id="dipep" name="dipep" size="30"
type="text" placeholder ="e.g. AA>1 ..." /></td>
                        </tr>
                        <tr>
                        <td class="yellowcell" align = "center"
><b>Display</b></td>

                        <td ><input type="checkbox" name="display"
id="aa_perc" value="aa_perc"  />Amino acid percentage 

                                    <input type="checkbox" name="display"
id="TMHMM" value="tm"  />TmHelix

                                        </tr>
                        </td>
                        </tr>

      </table>
    <tr><td></td><td> <input name="commit" type="submit"
onclick='storedata()'  value="Advanced Search" /></td></tr>
                  <br>
<tr><td><a href="./database_instructions" target = "_blank">Click here
for help</a>

</tr></td>

</table>
<p></p>

  </form>
  </div> <!-- end searchform div-->


$:{setup['menu']}
    </body>

</html>
```

APPENDIX I: WEBPY CODE

```
# This is a very simple web app that accepts jobs and submits them
# to the pipeline. Emailing the results to the submitter is handled
# by a cron job, since the job manager and/or sendmail configuration
# may prohibit the pipeline from sending email directly from a
# submitted job.
import os
import re
import sys
import json
```

```
import glob
import time
import subprocess
import cStringIO
import MySQLdb
from builtins import str

abspath = os.path.dirname(__file__)
sys.path.append(abspath)

from Config import *

#Add library paths to the system for use
sys.path.append(PROJECT_ROOT+'/pylib/')
sys.path.append(PROJECT_ROOT+'/pipeline/src/python/web/')
sys.path.append(PROJECT_ROOT+'/pipeline/src/beautifulsoup/')
sys.path.append(PROJECT_ROOT+'/pipeline/src/python/web/')
sys.path.append(OUTPUT_DIR)
os.chdir(abspath)

import web
import gpcrErrors

BBRC_HOME = PROJECT_ROOT

GENE_LIST_DESCRIPTION_FILE = 'descriptions.txt'

# Map of URLs to their handler classes.
urls = (
  '/', 'Index',
  '/results', 'Results',
  '/background', 'Background',
  '/instructions', 'Instructions',
  '/references', 'References',
  '/usefullinks', 'Links',
  '/database','Database',
  '/database_instructions','Database_Instructions',
  '/dbresults','Database_Results',
  '/dbfasta','Database_FASTA',
  '/contactus', 'Contact',
  '/getCSV', 'getCSV',
  '/get_dbCSV','get_dbCSV',
  '/get_dbFASTA', 'get_dbFASTA',
  '/get_dbCDHIT', 'get_dbCDHIT',
  '/processing', 'Processing',
  '/dbt','DBT'
)



# Load the template HTML files.
```

```
render =
web.template.render("%s/pipeline/src/templates/"%PROJECT_ROOT,globals=
{'str':str})

def getVars():
    vars = {'css': getCSS(),
            'banner': getBanner(),
            'navBar': getNavBar(),
            'favIco': getFavico(),
            'menu': getHTMLMenu(),
            'errors': '',
            'google_analytics_support': getGoogleAnalyticsBlock()}
    return vars

JOB_ID = time.time()

def getNextJobId():
  global JOB_ID
  JOB_ID += 1
  return JOB_ID

def getHTMLMenu():
  LINK_DIVIDER = ' | '
  return '\n'.join([
       '<hr/>',
       '<center>',
       LINK_DIVIDER.join([
           '<a href="%s">Home</a>'%URL_ROOT,
           '<a href="%s/background">Background</a>'%URL_ROOT,
           '<a href="%s/instructions">Instructions</a>'%URL_ROOT,
           '<a href="%s/references">References</a>'%URL_ROOT,
           '<a href="%s/usefullinks">Useful Links</a>'%URL_ROOT,
           '<a href="%s/database">Database</a>'%URL_ROOT,
           '<a href="%s/contactus">Contact Us</a>'%URL_ROOT
         ]),
       '<br/>This work is supported by the USDA-NIFA grant 2012-38422-
19910, and the NIMHD grant 2G12MD007592<br>',
       'The University of Texas at El Paso (UTEP), 500 W University
Ave. El Paso, TX 79968<br>',
       'Copyright 2018 GPCR, All Rights Reserved<br>',
       ' </center>',
       '<hr/>'
    ])

def getCSS():
    return "<link rel='stylesheet' href='" + URL_ROOT +
"/static/css/gpcr_style.css'>"

def getResultsCSS():
    return "<link rel='stylesheet' href='" + URL_ROOT +
"/static/css/gpcr_style_results.css'>\n\
```

```
            <link rel='stylesheet' href='" + URL_ROOT +
"/static/css/foundation.css'>"

def getDB_ResultsCSS():
    return "<link rel='stylesheet' href='" + URL_ROOT +
"/static/css/gpcr_db_style_results.css'>\n\
            <link rel='stylesheet' href='" + URL_ROOT +
"/static/css/foundation.css'>"


def getBanner():
    return "<img src='" + URL_ROOT +
"/static/img/Main_Banner_v5.png'>"

def getFavico():
    return "<link rel='shortcut icon' href='" + URL_ROOT +
"/static/img/fav.ico' type='image/x-icon'>"

def getNavBar():
    return "<div id='menu-bar'><ul>\
      <li><a href='" + URL_ROOT + "'>Home</a></li>\
         <li><a href='" + URL_ROOT + "/background'>Background</a></li>\
         <li><a href='" + URL_ROOT +
"/instructions'>Instructions</a></li>\
        <li><a href='" + URL_ROOT + "/references'>References</a></li>\
        <li><a href='" + URL_ROOT + "/usefullinks'>Useful
Links</a></li>\
      <li><a href='" + URL_ROOT + "/database'>Database</a><li>\
      <li><a href='" + URL_ROOT + "/contactus'>Contact us</a></li>\
      </ul></div>"

def getGoogleAnalyticsBlock():
    return '''
            <script>

(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||funct
ion(){

(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new
Date();a=s.createElement(o),

m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBe
fore(a,m)

})(window,document,'script','//www.google-
analytics.com/analytics.js','ga');
                                                    ga('create',
'UA-66562630-3', 'auto');

ga('send', 'pageview');

</script>
```

```
'''


class DBT:
   def GET(self,*args,**kws):
      import db_commands
      import sys
      print(sys.version)
      results = db_commands.classification('Non')
      vars=getVars()
      print(sys.version)
      return render.contactus(setup=vars)

class Contact:
   def GET(self,*args,**kws):
      vars = getVars()
      return render.contactus(setup=vars)

class Links:
   def GET(self,*args,**kws):
      vars = getVars()
      return render.usefullinks(setup=vars)

class Background:
   def GET(self,*args,**kws):
      vars = getVars()
      return render.background(setup=vars)


class Database:
   def GET(self):
      vars = getVars()
      return render.database(setup=vars)

   def POST(self,*args,**kws):
      print 'Within database post'
      inmap = web.input(search_term='',fastasequence='', qt='all',
a_organism='',a_pname = [''], a_gname =
[''],length_range='',a_length='', grafs = [''], iuphar =
[''],a_sub_fam=[''],a_sub_sub_fam = [''], a_sub_type = [''], aa='',
dipep='',display = [''],pro_type = [''])

      print inmap

      jobId = getNextJobId()
      fastasequence = inmap.fastasequence
      q=inmap.search_term
      qt= inmap.qt
      a_organism = inmap.a_organism
      a_pname = inmap.a_pname
      a_gname = inmap.a_gname
```

```
        length_range = inmap.length_range
        a_length = inmap.a_length
        grafs = inmap.grafs
        iuphar = inmap.iuphar
        a_sub_fam = inmap.a_sub_fam
        a_sub_sub_fam = inmap.a_sub_sub_fam
        a_sub_type = inmap.a_sub_type
        aa = inmap.aa
        dipep = inmap.dipep
        display = inmap.display
        pro_type = inmap.pro_type
        a_organism = [str(r) for r in a_organism.split(' OR ')]



        if fastasequence != '':
           #<PUT FASTA SEQUENCE FILE STUFF HERE>
           jobId = getNextJobId()
          # raise web.seeother("/dbresults?query=%s&qt=%s"%(q,qt))
           from datetime import date
           import io
           global contigMap
           try:
              directory = OUTPUT_DIR+'%s/'%jobId
              os.mkdir(directory)
              if len(fastasequence) > 0:
                 fastaSequence = fastasequence.splitlines()

              elif fastafile:
                 ffile = fastafile[0]
                 fastaSequence = ffile.splitlines()

              if len(fastaSequence) < 2:
                 vars = getVars()
                 vars['errors'] = gpcrErrors.getSubmissionError()
                 return render.index(setup=vars)

              print 'Before file creation'
              fastaSequences = shortenSequenceContig(fastaSequence)
              regExChars = list("|.^$*+?{}[]()\\")
              for n in fastaSequences:
                 for sc in regExChars:
                    if sc in list(n[1]):
                       vars = getVars()
                       vars['errors'] =
gpcrErrors.getSubmissionSCError(n[0])
                       return render.index(setup=vars)
                 sq = alt_seqtype_detect(n[1])

#                if (sq != 'P') or ((seq_type == 'D' or seq_type ==
'T') and sq != 'D'):
```

```
#                  print('In if')
#                  vars = getVars()
#                  vars['errors'] = gpcrErrors.getSequenceError()
#                  return render.index(setup=vars)
#               else:
#                  print('In else')

            with open(directory+'%s.Sequence.fasta'%jobId,'w+') as
resultFile:
               for item in fastaSequences:
                  resultFile.write(item[0]+'\n' + item[1]+'\n')
            resultFile.close()
            fastaFile = directory+"%s.Sequence.fasta"%jobId


            print 'Before parameter file'
#            createParameterFile(seq_type, gpcr_evalue,
min_len_protein_coding_regions, gpcr_fl_aa, gpcr_fl_pred_hel,
gpcr_max_pred_hel, gpcr_s_aa, gpcr_s_pred_hel, algos, directory,
jobId)
#            log =
open('%s%s.log'%(directory,('%s'%jobId).split('.')[0]), 'w')
#            log.write('Thanks for using the UTEP GPCR
Pipeline\nProcessing of your submission has started.\n')
#            log.close()
            os.environ["SHELL"]='/bin/bash'
#            comm = 'nohup %s/src/python/gpcrpipelineCL.py -in %s -
para %s%s.para -outdir %s > %s%s.nohup 2>&1
&'%(PIPELINE_ROOT,fastaFile, directory,jobId, directory, directory,
jobId)
            if sq == 'P':
               comm = 'nohup %s/bin/ncbi-blast-2.2.29/bin/blastp -
query %s -db %s/db/GPCR -evalue 0.05 -max_target_seqs 1 -outfmt \"6
qseqid sseqid stitle pident nident evalue\" -out %sblast_results.tab >
%s%s.nohup 2>&1 &'%(PIPELINE_ROOT,fastaFile,PROJECT_ROOT, directory,
directory, jobId)

            elif sq == 'D':
               comm = 'nohup %s/bin/ncbi-blast-2.2.29/bin/blastx -
query %s -db %s/db/GPCR -evalue 0.05 -max_target_seqs 1 -outfmt \"6
qseqid sseqid stitle pident nident evalue\" -out %sblast_results.tab >
%s%s.nohup 2>&1 &'%(PIPELINE_ROOT,fastaFile,PROJECT_ROOT, directory,
directory, jobId)
            cf = open('%sscript.sh'%(directory), 'w')
            cf.write(comm)
            cf.write('\nSTATUS=$?\necho $STATUS\nexit $STATUS\n')
            cf.write('echo \"Completed\"')
            cf.close()
            os.chmod('%sscript.sh'%directory, 0777)
            process = subprocess.Popen('/usr/bin/bash
%sscript.sh'%directory, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
```

```
            except:
               e = sys.exc_info()[0]
               vars = getVars()
               vars['error'] = gpcrErrors.getSubmissionError()
               print 'Within error'
               return render.database(setup=vars)
            time.sleep(5)
            raise web.seeother("/db_processing?jobId=%s"%str(jobId))
         elif q !='':
            raise web.seeother("/dbresults?query=%s&qt=%s"%(q,qt))
         elif a_organism != [''] or a_pname != [''] or a_gname != [''] or
a_length != '' or grafs != '' or iuphar != '' or a_sub_fam !=[''] or
a_sub_sub_fam != [''] or a_sub_type != [''] or aa != '' or dipep !=''
or pro_type != '':
            #print a_organism
            st_grafs = '+'.join(grafs)
            st_org = '+'.join(a_organism)
            st_pname = '+'.join(a_pname)
            st_gname = '+'.join(a_gname)
            st_iuphar = '+'.join(iuphar)
            st_display = '+'.join(display)
            st_pro_type = '+'.join(pro_type)
            st_a_sub_fam = '+'.join(a_sub_fam)
            st_a_sub_sub_fam = '+'.join(a_sub_sub_fam)
            st_a_sub_type = '+'.join(a_sub_type)

            raise
web.seeother("/dbresults?a_organism=%s&a_pname=%s&a_gname=%s&length_ra
nge=%s&a_length=%s&grafs=%s&iuphar=%s&a_sub_fam=%s&a_sub_sub_fam=%s&a_
sub_type=%s&aa=%s&dipep=%s&display=%s&pro_type=%s"%(st_org,st_pname,st
_gname,length_range,a_length,st_grafs,st_iuphar,st_a_sub_fam,st_a_sub_
sub_fam,st_a_sub_type,aa,dipep,st_display,st_pro_type))
         else:
            raise web.seeother("/dbresults?blank=Y")


class Database_Processing:
   def GET(self):
      vars = getVars()
      inmap = web.input(jobId=None)
      jobId = inmap.jobId
      print ('in processing')
      try:
         log =
open('%s%s/%s.log'%(OUTPUT_DIR,jobId,jobId.split('.')[0]),'r')
         lines = log.readlines()
         if re.match('Completed',lines[-1]):
            raise web.seeother('/results?jobId=%s'%jobId)
         else:
            status = 'The current status of jobId=%s \n'%jobId
            status = status + '<br>' + '<br>'.join(lines[1:])
```

```
            vars = getVars()
            vars['status'] = status
            return render.processing(setup=vars)
      except:
         return 'Error in Processing'

class Database_Results:
   def GET(self,*args,**kws):
      import db_commands
      vars = getVars()
      vars['css'] = getDB_ResultsCSS()

      inmap = web.input(blank='N',
query='',fastasequence='',qt='all',a_organism='',a_pname = '', a_gname
= '',length_range='',a_length='', grafs = '', iuphar = '',a_sub_fam =
'',a_sub_sub_fam = '', a_sub_type = '', aa='', dipep='',display =
'',pro_type = '')

      q = inmap.query
      qt = inmap.qt


      a_organism = inmap.a_organism.split()
      a_pname = inmap.a_pname.split()
      a_gname = inmap.a_gname.split()
      length_range = inmap.length_range
      a_length = inmap.a_length
      grafs = inmap.grafs.split()
      iuphar = inmap.iuphar.split()

      a_sub_fam = inmap.a_sub_fam.split()
      a_sub_sub_fam = inmap.a_sub_sub_fam.split()
      a_sub_type = inmap.a_sub_type.split()

      aa= inmap.aa.split()
      dipep = inmap.dipep.split()
      display = inmap.display.split()
      pro_type = inmap.pro_type.split()

      blank_search = inmap.blank
      if blank_search=='Y':
         results = db_commands.proid_search('')
         return render.db_results(setup=vars, rows=results)

      a_organism = [str(r) for r in a_organism]
      a_pname = [str(r) for r in a_pname]
      a_gname = [str(r) for r in a_gname]
      grafs = [str(r) for r in grafs]
      iuphar = [str(r) for r in iuphar]
      a_sub_fam = [str(r) for r in a_sub_fam]
      a_sub_sub_fam = [str(r) for r in a_sub_sub_fam]
      a_sub_type = [str(r) for r in a_sub_type]
```

```
      aa = [str(r) for r in aa]
      dipep = [str(r) for r in dipep]
      display = [str(r) for r in display]
      pro_type = [str(r) for r in pro_type]

      #cd_val = [str(r) for r in cd_val]
      print qt
      print "database result", a_sub_fam

      print 'Within database_results'

      if qt == 'organism':
         results = db_commands.organism_search(q)
      elif qt =='entry_name':
         results = db_commands.entryname_search(q)
      elif qt == 'protein_id':
         results = db_commands.proid_search(q)
      elif qt == 'classification':
         results = db_commands.classification(q)
      elif qt == 'amino_acid_composition':
         results = db_commands.aa_quick(q)
      elif qt == 'seq_length':
         results = db_commands.length_quick(q)
      elif qt == 'protein_name':
         results = db_commands.pname_search(q)
      elif qt == 'gene_name':
         results = db_commands.gname_search(q)

      elif a_organism != [] or a_pname != [] or a_gname != [] or
a_length != '' or grafs != [] or iuphar != [] or a_sub_fam != [] or
a_sub_sub_fam != [] or a_sub_type != [] or aa != [] or dipep != [] or
pro_type != [] or display != [] or pro_type !=[]:

         results =
db_commands.a_search(a_organism,a_pname,a_gname,length_range,a_length,
grafs,iuphar,a_sub_fam,a_sub_sub_fam,a_sub_type,aa,dipep,display,pro_t
ype)

      else:

         results = db_commands.proid_search('')
      # here create link
      if q != '':
         l =
"%s/get_dbCSV?query=%s&qt=%s"%(URL_ROOT,inmap.query,inmap.qt)
         fl =
"%s/dbfasta?query=%s&qt=%s"%(URL_ROOT,inmap.query,inmap.qt)
      else:
         l =
"%s/get_dbCSV?a_organism=%s&a_pname=%s&a_gname=%s&length_range=%s&a_le
ngth=%s&grafs=%s&iuphar=%s&a_sub_fam=%s&a_sub_sub_fam=%s&a_sub_type=%s
&aa=%s&dipep=%s&display=%s&pro_type=%s"%(URL_ROOT,inmap.a_organism,inm
```

```
ap.a_pname,inmap.a_gname,inmap.length_range,inmap.a_length,inmap.grafs
,inmap.iuphar,inmap.a_sub_fam,inmap.a_sub_sub_fam,inmap.a_sub_type,inm
ap.aa,inmap.dipep,inmap.display,inmap.pro_type)

         fl =
"%s/dbfasta?a_organism=%s&a_pname=%s&a_gname=%s&length_range=%s&a_leng
th=%s&grafs=%s&iuphar=%s&a_sub_fam=%s&a_sub_sub_fam=%s&a_sub_type=%s&a
a=%s&dipep=%s&display=%s&pro_type=%s"%(URL_ROOT,inmap.a_organism,inmap
.a_pname,inmap.a_gname,inmap.length_range,inmap.a_length,inmap.grafs,i
nmap.iuphar,inmap.a_sub_fam,inmap.a_sub_sub_fam,inmap.a_sub_type,inmap
.aa,inmap.dipep,inmap.display,inmap.pro_type)


      return render.db_results(setup=vars,
rows=results,link=l,fastalink=fl)


class Database_FASTA:
   def GET(self,*args,**kws):
      import db_commands

      vars = getVars()
      vars['css'] = getDB_ResultsCSS()

      inmap =
web.input(query='',cd_val='',search_term='',fastasequence='',
qt='all', a_organism='',a_pname ='', a_gname = '',
length_range='',a_length='', grafs = '', iuphar = '',a_sub_fam =
'',a_sub_sub_fam ='', a_sub_type = '', aa='', dipep='',display =
'',pro_type = '')

      print "Within db_fasta get"

      print inmap

      hid_vars = {'q' : inmap.query,'qt': inmap.qt,'a_organism' :
inmap.a_organism,'a_pname' : inmap.a_pname, 'a_gname': inmap.a_gname,
'length_range' : inmap.length_range,'a_length' :
inmap.a_length,'grafs' : inmap.grafs,'iuphar' :
inmap.iuphar,'a_sub_fam':inmap.a_sub_fam,
'a_sub_sub_fam':inmap.a_sub_sub_fam,
'a_sub_type':inmap.a_sub_type,'aa': inmap.aa,'dipep' :
inmap.dipep,'display' : inmap.display,'pro_type' :
inmap.pro_type,'cd_val' : inmap.cd_val}

      print hid_vars

      return render.db_fasta(setup=vars,hidden_vars= hid_vars)

   def POST(self,*args,**kws):
      import db_commands
      print 'Within database_fasta post'
```

```
      inmap = web.input(cd_val='',q='',fastasequence='', qt='all',
a_organism='',a_pname ='', a_gname = '', length_range='',a_length='',
grafs = '', iuphar = '',a_sub_fam ='',a_sub_sub_fam = '',a_sub_type =
'', aa='', dipep='',display = '',pro_type = '')


      jobId = getNextJobId()
      fastasequence = inmap.fastasequence
      q=inmap.q
      qt= inmap.qt
      cd_val = inmap.cd_val



      a_organism = inmap.a_organism.split()
      a_pname = inmap.a_pname.split()
      a_gname = inmap.a_gname.split()
      length_range = inmap.length_range
      a_length = inmap.a_length
      grafs = inmap.grafs.split()
      iuphar = inmap.iuphar.split()
      a_sub_fam = inmap.a_sub_fam.split()
      a_sub_sub_fam = inmap.a_sub_sub_fam.split()
      a_sub_type = inmap.a_sub_type.split()
      aa= inmap.aa.split()
      dipep = inmap.dipep.split()
      display = inmap.display.split()
      pro_type = inmap.pro_type.split()

      if qt == 'organism':
         results = db_commands.organism_search_fasta_db(q,jobId)
      elif qt =='entry_name':
         results = db_commands.entryname_search_fasta_db(q,jobId)
      elif qt == 'protein_id':
         results = db_commands.proid_search_fasta_db(q,jobId)
      elif qt == 'classification':
         results = db_commands.classification_fasta_db(q,jobId)
      elif qt == 'amino_acid_composition':
         results = db_commands.aa_quick_fasta_db(q,jobId)
      elif qt == 'seq_length':
         results = db_commands.length_quick_fasta_db(q,jobId)
      elif qt == 'protein_name':
         results = db_commands.pname_search_fasta_db(q,jobId)
      elif qt == 'gene_name':
         results = db_commands.gname_search_fasta_db(q,jobId)


      elif a_organism != [] or a_pname != [] or a_gname != [] or
a_length != '' or grafs != [] or iuphar != [] or a_sub_fam != [] or
a_sub_sub_fam != [] or a_sub_type != [] or aa != [] or dipep != [] or
pro_type != [] or display != [] or pro_type !=[]:
```

```
         results =
db_commands.a_search_fasta_db(a_organism,a_pname,a_gname,length_range,
a_length,grafs,iuphar,a_sub_fam,a_sub_sub_fam, a_sub_type,
aa,dipep,display,pro_type,jobId)


      else:

         results = db_commands.proid_search_fasta_db('')



      infile = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'r')

      toReturn_seq = ''.join(infile.readlines())

      import re

      word = ''

      if re.search('[a-zA-Z]+',cd_val) and cd_val.lower() !='full':

         vars = getVars()
         fl =
"%s/dbfasta?a_organism=%s&a_pname=%s&a_gname=%s&length_range=%s&a_leng
th=%s&grafs=%s&iuphar=%s&a_sub_fam=%s&a_sub_sub_fam=%s&a_sub_type=%s&a
a=%s&dipep=%s&display=%s&pro_type=%s"%(URL_ROOT,inmap.a_organism,inmap
.a_pname,inmap.a_gname,inmap.length_range,inmap.a_length,inmap.grafs,i
nmap.iuphar,inmap.a_sub_fam,inmap.a_sub_sub_fam,inmap.a_sub_type,inmap
.aa,inmap.dipep,inmap.display,inmap.pro_type)

         s = 'Please type the word: Full'
         return render.db_error(setup=vars,link = fl, s = s)


      elif cd_val.lower() == 'full':

         print 'db_fasta download result:'
         web.header('Content-Disposition', 'attachment;
filename=gpcrpendb_fasta%s.fasta'%getNextJobId())
         web.header('Content-Type', 'fasta')

         #return toReturn_seq
      elif float(cd_val) < 0.65 or float(cd_val) >1.0000000:

         vars = getVars()
         fl =
"%s/dbfasta?a_organism=%s&a_pname=%s&a_gname=%s&length_range=%s&a_leng
th=%s&grafs=%s&iuphar=%s&a_sub_fam=%s&a_sub_sub_fam=%s&a_sub_type=%s&a
a=%s&dipep=%s&display=%s&pro_type=%s"%(URL_ROOT,inmap.a_organism,inmap
.a_pname,inmap.a_gname,inmap.length_range,inmap.a_length,inmap.grafs,i
```

```
nmap.iuphar,inmap.a_sub_fam,inmap.a_sub_sub_fam,inmap.a_sub_type,inmap
.aa,inmap.dipep,inmap.display,inmap.pro_type)

         s = 'Please enter a valid number in the range from 0.65 to
1.0'
         return render.db_error(setup=vars,link = fl, s = s)
      elif float(cd_val) >= 0.65 :

         print '0.65 works'
         cd_name = cd_val.replace('.','_')
         print cd_name

         comm ='%s/bin/cdhit/cd-hit -i %s%s_query_seq.fasta -o
%s%s_out_%s.fasta -c
%s'%(PIPELINE_ROOT,OUTPUT_DIR,jobId,OUTPUT_DIR,jobId,cd_name,cd_val)
         process =
subprocess.Popen(comm,stdout=subprocess.PIPE,stderr=subprocess.PIPE,sh
ell=True)
         process.wait()
         print(process.stderr.readlines())
         print(process.stdout.readlines())

         infile =
open('%s%s_out_%s.fasta'%(OUTPUT_DIR,jobId,cd_name),'r')
         toReturn_seq = ''.join(infile.readlines())
         print 'db_fasta download result:'
         web.header('Content-Disposition', 'attachment;
filename=gpcr_cdhit_results_%s.fasta'%getNextJobId())
         web.header('Content-Type', 'fasta')
      return toReturn_seq
class get_dbCSV:
   def GET(self,*args,**kws):
      import db_commands
      vars = getVars()
      vars['css'] = getDB_ResultsCSS()

      inmap = web.input(cd_val='',blank='N',
query='',fastasequence='',qt='all',a_organism='',a_pname='',a_gname=''
,length_range='',a_length='', grafs = '', iuphar = '',a_sub_fam =
'',a_sub_sub_fam = '',a_sub_type = '', aa='', dipep='',display =
'',pro_type = '')
      q = inmap.query
      qt = inmap.qt
      a_organism = inmap.a_organism.split()
      a_pname = inmap.a_pname.split()
      a_gname = inmap.a_gname.split()
      length_range = inmap.length_range
      a_length = inmap.a_length
      grafs = inmap.grafs.split()
      iuphar = inmap.iuphar.split()
      aa= inmap.aa.split()
      a_sub_fam = inmap.a_sub_fam.split()
```

```
        a_sub_sub_fam = inmap.a_sub_sub_fam.split()
        a_sub_type = inmap.a_sub_type.split()
        dipep = inmap.dipep.split()
        display = inmap.display.split()
        pro_type = inmap.pro_type.split()
        cd_val = inmap.cd_val
        blank_search = inmap.blank
        if blank_search=='Y':
           results = db_commands.proid_search('')
           #return render.db_results(setup=vars, rows=results)

        a_organism = [str(r) for r in a_organism]
        a_pname = [str(r) for r in a_pname]
        a_gname = [str(r) for r in a_gname]

        grafs = [str(r) for r in grafs]
        iuphar = [str(r) for r in iuphar]
        a_sub_fam = [str(r) for r in a_sub_fam]
        a_sub_sub_fam = [str(r) for r in a_sub_sub_fam]
        a_sub_type = [str(r) for r in a_sub_type]
        aa = [str(r) for r in aa]
        dipep = [str(r) for r in dipep]
        display = [str(r) for r in display]
        pro_type = [str(r) for r in pro_type]
        #cd_val = [str(r) for r in cd_val]

        print 'Within getdbcsv_results'

        if qt == 'organism':
           results = db_commands.organism_search(q)
        elif qt =='entry_name':
           results = db_commands.entryname_search(q)
        elif qt == 'protein_id':
           results = db_commands.proid_search(q)
        elif qt == 'classification':
           results = db_commands.classification(q)
        elif qt == 'amino_acid_composition':
           results = db_commands.aa_quick(q)
        elif qt == 'seq_length':
           results = db_commands.length_quick(q)
        elif qt == 'protein_name':
           results = db_commands.pname_search(q)
        elif qt == 'gene_name':
           results = db_commands.gname_search(q)
        elif a_organism != [] or a_pname != [] or a_gname != [] or
a_length != '' or grafs != [] or iuphar != [] or a_sub_fam != [] or
a_sub_sub_fam != [] or a_sub_type != [] or aa != [] or dipep != [] or
pro_type != [] or display != [] or pro_type !=[]:

           results =
db_commands.a_search(a_organism,a_pname,a_gname,length_range,a_length,
```

```
grafs,iuphar,a_sub_fam,a_sub_sub_fam,
a_sub_type,aa,dipep,display,pro_type)

      else:

         results = db_commands.proid_search('')

      toReturn = ''
      toReturn = 'The number of proteins found: ' +
(str(results[0][0])) + '\n'

      toReturn = toReturn + 'Display: ' + str(results[0][1]) + '\n'

      toReturn = toReturn + 'Query terms: ' + str(results[0][2]) +
'\n'

      for r in results[1:]:
          b = []
          for rs in r:
             b.append(str(rs))
          toReturn = toReturn + ','.join(b) + '\n'

      #toReturn = toReturn + 'Download: <a href=\"/kbegum/get_dbCSV?\"
download target=\"_blank\">Results</a> \n'

      print 'db_CSV download result:'
      web.header('Content-Disposition', 'attachment;
filename=gpcrpendb_results_%s.csv'%getNextJobId())
      web.header('Content-Type', 'text/csv')

      return toReturn

class Database_Instructions:
   def GET(self,*args,**kws):
      vars = getVars()
      return render.database_instructions(setup=vars)

class Database_CDHIT:
   def GET(self,*args,**kws):
      vars = getVars()
      return render.db_cdhit(setup=vars)
```

APPENDIX J: CONVERTING AN INPUT TO MYSQL QUERY IN WEB-SERVER

```
# This code converts the input from the web-server of GPCR-PEnDB to
MySQL queries and accesses the database

cols_aa_tmhmm = 'SELECT
PROTEINS.PROTEIN_ID,PROTEINS.Name,PROTEINS.Protein_Name,PROTEINS.Gene_
Name,PROTEINS.Length,PROTEINS.PROTEIN_TYPE,Organism_v2.Genus,Organism_
```

```
v2.Species,Organism_v2.Common_name,round(PROTEINS.A*100,2),round(PROTE
INS.C*100,2),round(PROTEINS.D*100,2),round(PROTEINS.E*100,2),round(PRO
TEINS.F*100,2),round(PROTEINS.G*100,2),round(PROTEINS.H*100,2),round(P
ROTEINS.I*100,2),round(PROTEINS.K*100,2),round(PROTEINS.L*100,2),round
(PROTEINS.M*100,2),round(PROTEINS.N*100,2),round(PROTEINS.P*100,2),rou
nd(PROTEINS.Q*100,2),round(PROTEINS.R*100,2),round(PROTEINS.S*100,2),r
ound(PROTEINS.T*100,2),round(PROTEINS.V*100,2),round(PROTEINS.W*100,2)
,round(PROTEINS.Y*100,2),round
(PROTEINS.Other*100,2),TMHMM_Length.N_term,TMHMM_Length.TM1,TMHMM_Leng
th.IL1,TMHMM_Length.TM2,TMHMM_Length.OL1,TMHMM_Length.TM3,TMHMM_Length
.IL2,TMHMM_Length.TM4,TMHMM_Length.OL2,TMHMM_Length.TM5,TMHMM_Length.I
L3,TMHMM_Length.TM6,TMHMM_Length.OL3,TMHMM_Length.TM7,TMHMM_Length.C_t
erm FROM PROTEINS INNER JOIN Organism_v2 ON PROTEINS.Organism_ID =
Organism_v2.Organism_ID INNER JOIN GRAFS ON PROTEINS.GRAFS_ID =
GRAFS.GRAFS_ID INNER JOIN IUPHAR ON PROTEINS.IUPHAR_ID =
IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON PROTEINS.PROTEIN_ID =
AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length ON
TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID'

cols_grafs_iuphar_aa_tmhmm = 'SELECT
PROTEINS.PROTEIN_ID,PROTEINS.Name,PROTEINS.Protein_Name,PROTEINS.Gene_
Name,PROTEINS.Length,PROTEINS.PROTEIN_TYPE,Organism_v2.Genus,Organism_
v2.Species,Organism_v2.Common_name,GRAFS.GRAFS,
IUPHAR.IUPHAR,Ll_class.sub_family, Ll_class.s_sub_Family,
Ll_class.sub_type,PROTEINS.PDB_ID,round(PROTEINS.A*100,2),round(PROTEI
NS.C*100,2),round(PROTEINS.D*100,2),round(PROTEINS.E*100,2),round(PROT
EINS.F*100,2),round(PROTEINS.G*100,2),round(PROTEINS.H*100,2),round(PR
OTEINS.I*100,2),round(PROTEINS.K*100,2),round(PROTEINS.L*100,2),round(
PROTEINS.M*100,2),round(PROTEINS.N*100,2),round(PROTEINS.P*100,2),roun
d(PROTEINS.Q*100,2),round(PROTEINS.R*100,2),round(PROTEINS.S*100,2),ro
und(PROTEINS.T*100,2),round(PROTEINS.V*100,2),round(PROTEINS.W*100,2),
round(PROTEINS.Y*100,2),round
(PROTEINS.Other*100,2),TMHMM_Length.N_term,TMHMM_Length.TM1,TMHMM_Leng
th.IL1,TMHMM_Length.TM2,TMHMM_Length.OL1,TMHMM_Length.TM3,TMHMM_Length
.IL2,TMHMM_Length.TM4,TMHMM_Length.OL2,TMHMM_Length.TM5,TMHMM_Length.I
L3,TMHMM_Length.TM6,TMHMM_Length.OL3,TMHMM_Length.TM7,TMHMM_Length.C_t
erm FROM PROTEINS INNER JOIN Organism_v2 ON PROTEINS.Organism_ID =
Organism_v2.Organism_ID INNER JOIN GRAFS ON PROTEINS.GRAFS_ID =
GRAFS.GRAFS_ID INNER JOIN IUPHAR ON PROTEINS.IUPHAR_ID =
IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON PROTEINS.PROTEIN_ID =
AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length ON
TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID INNER JOIN Ll_class ON
PROTEINS.PROTEIN_ID = Ll_class.PROTEIN_ID'


cols_grafs_iuphar= 'SELECT
PROTEINS.PROTEIN_ID,PROTEINS.Name,PROTEINS.Protein_Name,PROTEINS.Gene_
Name,PROTEINS.Length,PROTEINS.PROTEIN_TYPE,Organism_v2.Genus,Organism_
v2.Species,Organism_v2.Common_name,GRAFS.GRAFS,
IUPHAR.IUPHAR,Ll_class.sub_family, Ll_class.s_sub_family,
Ll_class.sub_type,PROTEINS.PDB_ID FROM PROTEINS INNER JOIN Organism_v2
ON PROTEINS.Organism_ID = Organism_v2.Organism_ID INNER JOIN GRAFS ON
```

```
PROTEINS.GRAFS_ID = GRAFS.GRAFS_ID INNER JOIN IUPHAR ON
PROTEINS.IUPHAR_ID = IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON
PROTEINS.PROTEIN_ID = AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length
ON TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID INNER JOIN Ll_class
ON PROTEINS.PROTEIN_ID = Ll_class.PROTEIN_ID'

cols_aa = 'SELECT
PROTEINS.PROTEIN_ID,PROTEINS.Name,PROTEINS.Protein_Name,PROTEINS.Gene_
Name,PROTEINS.Length,PROTEINS.PROTEIN_TYPE,Organism_v2.Genus,Organism_
v2.Species,Organism_v2.Common_name,GRAFS.GRAFS,IUPHAR.IUPHAR,Ll_class.
sub_family, Ll_class.s_sub_family,
Ll_class.sub_type,PROTEINS.PDB_ID,round(PROTEINS.A*100,2),round(PROTEI
NS.C*100,2),round(PROTEINS.D*100,2),round(PROTEINS.E*100,2),round(PROT
EINS.F*100,2),round(PROTEINS.G*100,2),round(PROTEINS.H*100,2),round(PR
OTEINS.I*100,2),round(PROTEINS.K*100,2),round(PROTEINS.L*100,2),round(
PROTEINS.M*100,2),round(PROTEINS.N*100,2),round(PROTEINS.P*100,2),roun
d(PROTEINS.Q*100,2),round(PROTEINS.R*100,2),round(PROTEINS.S*100,2),ro
und(PROTEINS.T*100,2),round(PROTEINS.V*100,2),round(PROTEINS.W*100,2),
round(PROTEINS.Y*100,2),round(PROTEINS.Other*100,2) FROM PROTEINS
INNER JOIN Organism_v2 ON PROTEINS.Organism_ID =
Organism_v2.Organism_ID INNER JOIN GRAFS ON PROTEINS.GRAFS_ID =
GRAFS.GRAFS_ID INNER JOIN IUPHAR ON PROTEINS.IUPHAR_ID =
IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON PROTEINS.PROTEIN_ID =
AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length ON
TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID INNER JOIN Ll_class ON
PROTEINS.PROTEIN_ID = Ll_class.PROTEIN_ID'

cols_tmhmm = 'SELECT
PROTEINS.PROTEIN_ID,PROTEINS.Name,PROTEINS.Protein_Name,PROTEINS.Gene_
Name,PROTEINS.Length,PROTEINS.PROTEIN_TYPE,Organism_v2.Genus,Organism_
v2.Species,Organism_v2.Common_name,GRAFS.GRAFS,IUPHAR.IUPHAR,Ll_class.
sub_family, Ll_class.s_sub_family,
Ll_class.sub_type,PROTEINS.PDB_ID,TMHMM_Length.N_term,TMHMM_Length.TM1
,TMHMM_Length.IL1,TMHMM_Length.TM2,TMHMM_Length.OL1,TMHMM_Length.TM3,T
MHMM_Length.IL2,TMHMM_Length.TM4,TMHMM_Length.OL2,TMHMM_Length.TM5,TMH
MM_Length.IL3,TMHMM_Length.TM6,TMHMM_Length.OL3,TMHMM_Length.TM7,TMHMM
_Length.C_term FROM PROTEINS INNER JOIN Organism_v2 ON
PROTEINS.Organism_ID = Organism_v2.Organism_ID INNER JOIN GRAFS ON
PROTEINS.GRAFS_ID = GRAFS.GRAFS_ID INNER JOIN IUPHAR ON
PROTEINS.IUPHAR_ID = IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON
PROTEINS.PROTEIN_ID = AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length
ON TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID INNER JOIN Ll_class
ON PROTEINS.PROTEIN_ID = Ll_class.PROTEIN_ID'


def
a_search(a_organism,length_range,a_length,grafs,iuphar,aa,dipep,displa
y,pro_type):

   import MySQLdb
   from Config import *
```

```
   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


#=====================================================================
======
#
# Quick Search
#
#=====================================================================
======

#--------------Organism Search----------------------------------------
--------------------------#
def organism_search(q):

   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


   # execute SQL select statement

   query = '%s '%(cols_aa_tmhmm) + 'WHERE Organism_v2.Common_name LIKE
\"%{0}%\" OR Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species
LIKE \"%{0}%\";'.format(q)

   print query
   cursor.execute(query)
   d_h = 'Amino acid percentages & TMHMM predicted lengths'
   q_h = 'Organism name = ' + q
   # commit your changes
   db.commit()

   numrows = int(cursor.rowcount)
   toReturn = [(numrows,q_h,d_h),('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

   # get and display one row at a time.
   for x in range(0,numrows):
```

```
      row = cursor.fetchone()
      toReturn.append(row)
   #print toReturn
   return toReturn

#----------------------------------------Entry Name----------------
-----------------------------------#
def entryname_search(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
   toReturn = []

   # execute SQL select statement
   query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.Name LIKE
\"%{0}%\";'.format(q)

   cursor.execute(query)

   d_h = 'Amino acid percentages & TMHMM predicted lengths'
   q_h = 'Entry name = ' + q

   # commit your changes
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn = [(numrows,q_h,d_h),('ID','Entry Name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

   numrows = int(cursor.rowcount)

   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)

   return toReturn



#----------------------------------------Protein Name--------------
-----------------------------------#
def pname_search(q):
   import MySQLdb
```

```
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
   toReturn = []

   # execute SQL select statement
   query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.Protein_Name LIKE
\"%{0}%\";'.format(q)

   cursor.execute(query)

   d_h = 'Amino acid percentages & TMHMM predicted lengths'
   q_h = 'Protein name = ' + q

   # commit your changes
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn = [(numrows,q_h,d_h),('ID','Entry Name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

   numrows = int(cursor.rowcount)

   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)

   return toReturn


#----------------------------------------Gene Name----------------
----------------------------------#
def gname_search(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
   toReturn = []

   # execute SQL select statement
```

```
   query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.Gene_Name LIKE
\"%{0}%\";'.format(q)

   cursor.execute(query)

   d_h = 'Amino acid percentages & TMHMM predicted lengths'
   q_h = 'Gene name = ' + q

   # commit your changes
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn = [(numrows,q_h,d_h),('ID','Entry Name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

   numrows = int(cursor.rowcount)

   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)

   return toReturn


#-------------------------------------------------Uniprot ID--------
----------------------------------------#
def proid_search(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)


   cursor = db.cursor()


   # execute SQL select statement
   query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.PROTEIN_ID LIKE
\"%{0}%\";'.format(q)

   cursor.execute(query)
   if q == '':
      d_h = 'Amino acid percentages & TMHMM predicted lengths'

      q_h = 'All'
   else:
```

```
      q_h = 'UniProtKB ID = ' + q
      d_h = 'Amino acid percentages & TMHMM predicted lengths'

   # commit your changes
   db.commit()

   numrows = int(cursor.rowcount)

   toReturn = [(numrows,q_h,d_h),('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)
   print query
   return toReturn

#----------------------------Classification----------------------------
----------------------------#
def classification(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)


   cursor = db.cursor()
   # execute SQL select statement

#Query for non GPCRs
   if q == 'Non' or q == 'non' or q == 'None' or q == 'none':
      query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.PROTEIN_TYPE
="CN";'
      cursor.execute(query)
      db.commit()
      q_h = 'Classification = ' + q
      d_h = 'Amino acid percentages & TMHMM predicted lengths'

      numrows = int(cursor.rowcount)
      toReturn = [(numrows,q_h,d_h),('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
```

```
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn.append(row)
      print query

      return toReturn
   else:
#Query for GPCR classifications
      query = '%s '%(cols_grafs_iuphar_aa_tmhmm) + 'WHERE
IUPHAR.IUPHAR LIKE \"%{0}%\" OR GRAFS.GRAFS LIKE \"%{0}%\";'.format(q)
  # print query
      cursor.execute(query)

   # commit your changes
      db.commit()
      q_h = 'Classification = ' + q
      d_h = 'Amino acid percentages & TMHMM predicted lengths'

      numrows = int(cursor.rowcount)


      #toReturn = []
      #toReturn.append((numrows))


      toReturn = [(numrows,q_h,d_h),('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-type',
'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V
','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
     # toReturn = toReturn + toReturn1
      print toReturn
   # get and display one row at a time.
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn.append(row)
     # print toReturn[0]
      return toReturn


#----------------------------AA percentage-----------------------------
--------------------------#

def aa_quick(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)
```

```
   cursor = db.cursor()

   # execute SQL select statement
   query = '%s '%(cols_aa_tmhmm) + 'WHERE
PROTEINS.\"{0}\/100";'.format(q)


   query = query.replace('"','')
   query = query.replace('\\','')
   print query
  # print query
   cursor.execute(query)
   q_h = 'Amino Acid, ' + q
   d_h = 'TMHMM predicted lengths'


   # commit your changes
   db.commit()

   numrows = int(cursor.rowcount)
   toReturn = [(numrows,q_h,d_h),('ID','Entry Name','Protein
name','Gene','Length','Type','Genus','Species','Common
name','A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S',
'T','V','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
   toReturn
   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)

   return toReturn


#----------------------------Length-----------------------------------
----------------------------#

def length_quick(q):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


   # execute SQL select statement
```

```
   query = '%s '%(cols_aa_tmhmm) + 'WHERE PROTEINS.Length
\"{0}\";'.format(q)

   query = query.replace('"','')
   query = query.replace('\\','')
   cursor.execute(query)
   print query
   q_h = 'Length ' + q
   d_h = 'Amino acid percentages & TMHMM predicted lengths'

   # commit your changes
   db.commit()

   numrows = int(cursor.rowcount)

   toReturn = [(numrows,q_h,d_h),('ID','Entry Name','Protein
name','Gene','Length','Type','Genus','Species','Common
name','A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S',
'T','V','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

   # get and display one row at a time.
   for x in range(0,numrows):
      row = cursor.fetchone()
      toReturn.append(row)

   return toReturn



#====================================================================
======
#
# advanced SEARCH
#
#====================================================================
======
def
a_search(a_organism,a_pname,a_gname,length_range,a_length,grafs,iuphar
,a_sub_fam,a_sub_sub_fam,a_sub_type,aa,dipep,display,pro_type):

   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()

# If all the search term are null except "Display" and "Category"
```

```
   if (display != [] or pro_type != [])  and (len(a_organism) == 0 )
and (len(a_pname) == 0 ) and (len(a_gname) == 0) and a_length == ''
and (len(iuphar) == 0 ) and (len(grafs) == 0 ) and (len(a_sub_fam)
==0) and (len(a_sub_sub_fam) == 0) and (len(a_sub_type) == 0) and
(len(aa) == 0) and (len(dipep) == 0 ):
      if display ==[] and pro_type != []:

         d_h = 'GRAFS & IUPHAR'
         toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name', 'GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID')]

         if len(pro_type) == 1:

            if pro_type[0] == 'cgf_type':
            #Every box is empty, only pro_type = cg_type
               query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF";'
               q_h = 'Confirmed GPCR fragments'

            elif pro_type[0] == 'cg_type':
            #Every box is empty only pro_type = cn_type

             query = '%s '%(cols_grafs_iuphar)+ ' WHERE
PROTEINS.PROTEIN_TYPE = "CG";'
               q_h = 'Confirmed GPCRs'
            elif pro_type[0] == 'cn_type':
               query = '%s '%(cols_grafs_iuphar) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN";'
               q_h = 'Confirmed Non-GPCRs'
            elif pro_type[0] == 'cnt_type':
               query = '%s '%(cols_grafs_iuphar) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT";'
               q_h = 'Confirmed Non-GPCRs(transmembrane)'
         elif len(pro_type) == 2:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type':

                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF";'
                  q_h = 'Confirmed GPCRs & fragments'

               elif pro_type[1] == 'cn_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN";'
                  q_h = 'Confirmed GPCRs & Non-GPCRs'
               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_grafs_iuphar) + 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT";'
                  q_h = 'Confirmed GPCRs & Non-GPCRs(transmembrane)'
```

```
            elif pro_type[0] == 'cgf_type':
               if pro_type[1] == 'cn_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN";'
                  q_h = 'Confirmed GPCR fragments & Non-GPCRs'
               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                  q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane)'
            else:
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT";'
               q_h = 'Confirmed Non-GPCRs including transmembrane'
         elif len(pro_type) == 3:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN";'
                  q_h = 'Confirmed GPCRs, GPCR fragments & Non-GPCRs'
               elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                  q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane)'
               elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT";'
                  q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane'
            else:
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
               q_h = 'Confirmed GPCR fragments & Non-GPCRs including
transmembrane'
      else:
            query = '%s '%(cols_grafs_iuphar)+ ';'
            q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane'

      elif display != [] and pro_type == []:


         if display[0] == 'aa_perc' and len(display) == 1:
```

```
            #Every box is empty except display  = aa
            query = '%s '%(cols_aa) + ' ;'
            d_h = 'GRAFS,IUPHAR & Amino acid percentages'
            q_h = 'None'

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others')]

         elif display[0] == 'tm' and len(display) == 1:
         #Every box is empty except display = tm
            query = '%s '%(cols_tmhmm) + ' ;'
            d_h = 'GRAFS, IUPHAR & TMHMM predicted lengths'
            q_h = 'None'

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
         else:
         #Every box is empty except show 'tm' and 'aa_perc'
            query = '%s '%(cols_grafs_iuphar_aa_tmhmm) + ';'
            d_h = 'GRAFS, IUPHAR, Amino acid percentages and TMHMM
predicted lengths'
            q_h = 'None'

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

##### Start editing here

      elif display !=[] and pro_type != []:
         if display[0] == 'tm' and len(display) == 1:

            d_h = 'GRAFS, IUPHAR & TMHMM predicted lengths'
            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

            if len(pro_type) == 1:
```

```
         #Every box is empty except display = 'tm' and pro_type =
'cn_type'
               if pro_type[0] == 'cgf_type':
                  query = '%s '%(cols_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" ;'
                  q_h = 'Confirmed GPCR fragments'
               elif pro_type[0] == 'cg_type':
            #Every box is empty except display = 'tm' and pro_type =
'cg_type'
                  query = '%s '%(cols_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CG" ;'
                  q_h = 'Confirmed GPCRs'
               elif pro_type[0] == 'cn_type':
            #Every box is empty except display = 'tm' and pro_type =
'cn_type'
                  query = '%s '%(cols_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CN" ;'
                  q_h = 'Confirmed Non-GPCRs'
               else:
                  query = '%s '%(cols_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CNT" ;'
                  q_h = 'Confirmed Non-GPCRs(transmembrane)'

            elif len(pro_type) == 2:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type':

                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF";'
                     q_h = 'Confirmed GPCRs & GPCR fragments'

                  elif pro_type[1] == 'cn_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR & Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCRs & Non-
GPCRs(transmembrane)'
               elif pro_type[0] == 'cgf_type':
                  if pro_type[1] == 'cn_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR fragments & Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane)'
               else:
```

```
                    query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                    q_h = 'Confirmed Non-GPCRs including transmembrane'

            elif len(pro_type) == 3:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN";'
                     q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs'
                  elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                     q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane)'
                  elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane'
               else:
                  query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                  q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane'

         else:
            #Every box is empty only pro type is cg_type and cn_type
both

               query = '%s '%(cols_tmhmm)+ ';'
               q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane'


         elif display[0] == 'aa_perc' and len(display) == 1:
            d_h = 'GRAFS, IUPHAR & Amino acid percentages'
            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others')]
```

```
            if len(pro_type) == 1:
          #Every box is empty except display = 'tm' and pro_type =
'cn_type'
               if pro_type[0] == 'cgf_type':
                  query = '%s '%(cols_aa) +'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" ;'
                  q_h = 'Confirmed GPCR fragments'
               elif pro_type[0] == 'cg_type':
             #Every box is empty except display = 'tm' and pro_type =
'cg_type'
                  query = '%s '%(cols_aa) +'WHERE
PROTEINS.PROTEIN_TYPE = "CG" ;'
                  q_h = 'Confirmed GPCRs'
               elif pro_type[0] == 'cn_type':
             #Every box is empty except display = 'tm' and pro_type =
'cn_type'
                  query = '%s '%(cols_aa) +'WHERE
PROTEINS.PROTEIN_TYPE = "CN" ;'
                  q_h = 'Confirmed Non-GPCRs'
               else:
                  query = '%s '%(cols_aa) +'WHERE
PROTEINS.PROTEIN_TYPE = "CNT" ;'
                  q_h = 'Confirmed Non-GPCRs(transmembrane)'

            elif len(pro_type) == 2:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type':

                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF";'
                     q_h = 'Confirmed GPCRs & GPCR fragments'

                  elif pro_type[1] == 'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR & Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCRs & Non-
GPCRs(transmembrane)'
               elif pro_type[0] == 'cgf_type':
                  if pro_type[1] == 'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR fragments & Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane)'
               else:
```

```
                    query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                    q_h = 'Confirmed Non-GPCRs including transmembrane'

            elif len(pro_type) == 3:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN";'
                     q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs'
                  elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                     q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane)'
                  elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane'
               else:
                  query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                  q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane'

         else:
            #Every box is empty only pro type is cg_type and cn_type
both

               query = '%s '%(cols_aa)+ ';'
               q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane'


         elif len(display) == 2:
            d_h = 'GRAFS, IUPHAR, Amino acid percentages & TMHMM
predicted lengths'
            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others','N-
```

```
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

            if len(pro_type) == 1:
          #Every box is empty except display = 'tm' and pro_type =
'cn_type'
               if pro_type[0] == 'cgf_type':
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" ;'
                  q_h = 'Confirmed GPCR fragments'
               elif pro_type[0] == 'cg_type':
            #Every box is empty except display = 'tm' and pro_type =
'cg_type'
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CG" ;'
                  q_h = 'Confirmed GPCRs'
               elif pro_type[0] == 'cn_type':
            #Every box is empty except display = 'tm' and pro_type =
'cn_type'
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CN" ;'
                  q_h = 'Confirmed Non-GPCRs'
               else:
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) +'WHERE
PROTEINS.PROTEIN_TYPE = "CNT" ;'
                  q_h = 'Confirmed Non-GPCRs(transmembrane)'

            elif len(pro_type) == 2:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type':

                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF";'
                     q_h = 'Confirmed GPCRs and GPCR fragments'

                  elif pro_type[1] == 'cn_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR and Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                     q_h = 'Confirmed GPCRs and Non-
GPCRs(transmembrane)'
               elif pro_type[0] == 'cgf_type':
                  if pro_type[1] == 'cn_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN";'
                     q_h = 'Confirmed GPCR fragments & Non-GPCRs'
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT";'
```

```
                    q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane)'
                else:
                    query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT";'
                    q_h = 'Confirmed Non-GPCRs including transmembrane'
             #print pro_type
             elif len(pro_type) == 3:
                if pro_type[0] == 'cg_type':
                   if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                      query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN";'
                      q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs'
                   elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                      query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                      q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane)'
                   elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                      query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT";'
                      q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane'
                else:
                   query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT";'
                   q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane'

          else:
             #Every box is empty only pro type is cg_type and cn_type
both
                query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ ';'
                q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane'



#-------------------when there is a query term-----------------------
-------------------------------------#

   elif (display != [] or pro_type != [])  and ((len(a_organism) != 0)
or (a_pname != '') or (a_gname != '') or (a_length != '') or
(len(iuphar) != 0) or (len(grafs) != 0) or (len(a_sub_fam) !=0) or
```

```
(len(a_sub_sub_fam) != 0) or (len(a_sub_type) != 0) or (len(aa) != 0 )
or (len(dipep) != 0 )):
      if display ==[] and pro_type != []:
         d_h = 'GRAFS & IUPHAR'
         toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name', 'GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID')]

         if len(pro_type) == 1:

            if pro_type[0] == 'cgf_type':
            #Every box is empty, only pro_type = cg_type
               query = '%s '%(cols_grafs_iuphar)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" AND ('
               q_h = 'Confirmed GPCR fragments + '

            elif pro_type[0] == 'cg_type':
            #Every box is empty only pro_type = cn_type

             query = '%s '%(cols_grafs_iuphar)+ ' WHERE
PROTEINS.PROTEIN_TYPE = "CG" AND ('
               q_h = 'Confirmed GPCRs + '
            elif pro_type[0] == 'cn_type':
               query = '%s '%(cols_grafs_iuphar) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN" AND ('
               q_h = 'Confirmed Non-GPCRs + '
            elif pro_type[0] == 'cnt_type':
               query = '%s '%(cols_grafs_iuphar) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT" AND ('
               q_h = 'Confirmed Non-GPCRs(transmembrane) + '
         elif len(pro_type) == 2:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type':

                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF") AND ('
                  q_h = 'Confirmed GPCRs & fragments + '

               elif pro_type[1] == 'cn_type':
                  query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN") AND ('
                  q_h = 'Confirmed GPCRs and Non-GPCRs + '
               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_grafs_iuphar) + 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                  q_h = 'Confirmed GPCRs & Non-GPCRs(transmembrane) +
'

            elif pro_type[0] == 'cgf_type':
               if pro_type[1] == 'cn_type':
```

```
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN") AND ('
                q_h = 'Confirmed GPCR fragments & Non-GPCRs + '
             elif pro_type[1] == 'cnt_type':
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane) + '
          else:
              query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
              q_h = 'Confirmed Non-GPCRs including transmembrane + '
       elif len(pro_type) == 3:
          if pro_type[0] == 'cg_type':
             if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN") AND ('
                q_h = 'Confirmed GPCRs, GPCR fragments & Non-GPCRs +
'
             elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane) + '
             elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane + '
          else:
              query = '%s '%(cols_grafs_iuphar)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
              q_h = 'Confirmed GPCR fragments & Non-GPCRs including
transmembrane + '
      else:
           query = '%s '%(cols_grafs_iuphar)+ 'WHERE ('
           q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane + '


     elif display != [] and pro_type == []:
        if display[0] == 'aa_perc' and len(display) == 1:
           #Every box is empty except display  =
           query = '%s '%(cols_aa) + ' WHERE ('
           d_h = 'GRAFS, IUPHAR & Amino acid percentages'
```

```
            q_h = 'Confirmed Non-GPCRs + '

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others')]
         elif display[0] == 'tm' and len(display) == 1:
         #Every box is empty except display = tm
            query = '%s'%(cols_tmhmm) + ' WHERE ('
            d_h = 'GRAFS, IUPHAR & TMHMM predicted lengths'
            q_h = 'Confirmed GPCRs and Non-GPCRs + '

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]
         else:
         #Every box is empty except show 'tm' and 'aa_perc'
            query = '%s'%(cols_grafs_iuphar_aa_tmhmm) +' WHERE ('
            d_h = 'GRAFS, IUPHAR, Amino acid percentages and TMHMM
predicted lengths'
            q_h = 'Confirmed GPCRs and Non-GPCRs + '

            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

      elif display !=[] and pro_type != []:
         if display[0] == 'tm' and len(display) == 1:
            d_h = 'GRAFS, IUPHAR & TMHMM predicted lengths'
            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

            if len(pro_type) == 1:

               if pro_type[0] == 'cgf_type':
            #Every box is empty, only pro_type = cg_type
                  query = '%s '%(cols_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" AND ('
                  q_h = 'Confirmed GPCR fragments + '
```

```
            elif pro_type[0] == 'cg_type':
         #Every box is empty only pro_type = cn_type

             query = '%s '%(cols_tmhmm)+ ' WHERE
PROTEINS.PROTEIN_TYPE = "CG" AND ('
               q_h = 'Confirmed GPCRs + '
            elif pro_type[0] == 'cn_type':
               query = '%s '%(cols_tmhmm) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN" AND ('
               q_h = 'Confirmed Non-GPCRs + '
            elif pro_type[0] == 'cnt_type':
               query = '%s '%(cols_tmhmm) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT" AND ('
               q_h = 'Confirmed Non-GPCRs(transmembrane) + '
         elif len(pro_type) == 2:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type':

                  query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF") AND ('
                  q_h = 'Confirmed GPCRs & fragments + '

               elif pro_type[1] == 'cn_type':
                  query = '%s '%(cols_tmhm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN") AND ('
                  q_h = 'Confirmed GPCRs and Non-GPCRs + '
               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_tmhmm) + 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                  q_h = 'Confirmed GPCRs & Non-GPCRs(transmembrane)
+ '

            elif pro_type[0] == 'cgf_type':
               if pro_type[1] == 'cn_type':
                  query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN") AND ('
                  q_h = 'Confirmed GPCR fragments & Non-GPCRs + '
               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                  q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane) + '
            else:
               query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
               q_h = 'Confirmed Non-GPCRs including transmembrane +
'
         elif len(pro_type) == 3:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
```

```
                    query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN") AND ('
                    q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs + '
                elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                    query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                    q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane) + '
                elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                    query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                    q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane + '
             else:
                query = '%s '%(cols_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane + '
         else:
              query = '%s '%(cols_tmhmm)+ 'WHERE ('
              q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane + '

        elif display[0] == 'aa_perc' and len(display) == 1:
           d_h = 'GRAFS, IUPHAR & Amino acid percentages'
           toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others')]

           if len(pro_type) == 1:

              if pro_type[0] == 'cgf_type':
           #Every box is empty, only pro_type = cg_type
                query = '%s '%(cols_aa)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" AND ('
                q_h = 'Confirmed GPCR fragments + '

              elif pro_type[0] == 'cg_type':
           #Every box is empty only pro_type = cn_type

               query = '%s '%(cols_aa)+ ' WHERE PROTEINS.PROTEIN_TYPE
= "CG" AND ('
                q_h = 'Confirmed GPCRs + '
```

```
               elif pro_type[0] == 'cn_type':
                  query = '%s '%(cols_aa) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN" AND ('
                  q_h = 'Confirmed Non-GPCRs + '
               elif pro_type[0] == 'cnt_type':
                  query = '%s '%(cols_aa) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT" AND ('
                  q_h = 'Confirmed Non-GPCRs(transmembrane) + '
            elif len(pro_type) == 2:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type':

                     query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF") AND ('
                     q_h = 'Confirmed GPCRs & fragments + '

                  elif pro_type[1] == 'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN") AND ('
                     q_h = 'Confirmed GPCRs and Non-GPCRs + '
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_aa) + 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                     q_h = 'Confirmed GPCRs & Non-GPCRs(transmembrane)
+ '

               elif pro_type[0] == 'cgf_type':
                  if pro_type[1] == 'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN") AND ('
                     q_h = 'Confirmed GPCR fragments & Non-GPCRs + '
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                     q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane) + '
               else:
                  query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                  q_h = 'Confirmed Non-GPCRs including transmembrane +
'
            elif len(pro_type) == 3:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN") AND ('
                     q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs + '
                  elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
```

```
                    query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                    q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane) + '
                elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                    query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                    q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane + '
              else:
                 query = '%s '%(cols_aa)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                 q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane + '
         else:
              query = '%s '%(cols_aa)+ 'WHERE ('
              q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane + '



        elif len(display) == 2:
           d_h = 'GRAFS, IUPHAR, Amino acid percentages & TMHMM
predicted lengths'
            toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID','A','C','D','E','F','G','H','I','K','L','M','N','P','Q'
,'R','S','T','V','W','Y ','Others','N-
term','H1','IL1','H2','OL1','H3','IL2','H4','OL2','H5','IL3','H6','OL3
','H7','C-term')]

            if len(pro_type) == 1:

               if pro_type[0] == 'cgf_type':
            #Every box is empty, only pro_type = cg_type
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" AND ('
                  q_h = 'Confirmed GPCR fragments + '

               elif pro_type[0] == 'cg_type':
            #Every box is empty only pro_type = cn_type

                query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ ' WHERE
PROTEINS.PROTEIN_TYPE = "CG" AND ('
                  q_h = 'Confirmed GPCRs + '
               elif pro_type[0] == 'cn_type':
```

```
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN" AND ('
                  q_h = 'Confirmed Non-GPCRs + '
               elif pro_type[0] == 'cnt_type':
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT" AND ('
                  q_h = 'Confirmed Non-GPCRs(transmembrane) + '
            elif len(pro_type) == 2:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type':

                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF")
AND ('
                     q_h = 'Confirmed GPCRs & fragments + '

                  elif pro_type[1] == 'cn_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN")
AND ('
                     q_h = 'Confirmed GPCRs and Non-GPCRs + '
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm) +
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT")
AND ('
                     q_h = 'Confirmed GPCRs & Non-GPCRs(transmembrane)
+ '

               elif pro_type[0] == 'cgf_type':
                  if pro_type[1] == 'cn_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN")
AND ('
                     q_h = 'Confirmed GPCR fragments & Non-GPCRs + '
                  elif pro_type[1] == 'cnt_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT")
AND ('
                     q_h = 'Confirmed GPCR fragments & Non-
GPCRs(transmembrane) + '
               else:
                  query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                  q_h = 'Confirmed Non-GPCRs including transmembrane +
'
            elif len(pro_type) == 3:
               if pro_type[0] == 'cg_type':
                  if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                     query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF"
OR PROTEINS.PROTEIN_TYPE = "CN") AND ('
```

```
                    q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs + '
                elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                    query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF"
OR PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                    q_h = 'Confirmed GPCRs, GPCR fragments & Non-
GPCRs(transmembrane) + '
                elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                    query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+
'WHERE (PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT") AND ('
                    q_h = 'Confirmed GPCRs & Non-GPCRs including
transmembrane + '
            else:
                query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('
                q_h = 'Confirmed GPCR fragments & Non-GPCRs
including transmembrane + '
        else:
             query = '%s '%(cols_grafs_iuphar_aa_tmhmm)+ 'WHERE ('
             q_h = 'Confirmed GPCRs including fragments & Non-GPCRs
including transmembrane + '


   else:

      query = '%s '%(cols_grafs_iuphar) + 'WHERE ('
      d_h = 'GRAFS & IUPHAR'
      q_h = ''

      toReturn = [('ID','Entry name','Protein
name','Gene','Length','Type','Genus','Species','Common name','GRAFS
(family)','IUPHAR (family)','Sub-family','Sub-sub-family','Sub-
type','PDB_ID')]




#----------------------------Organism part----------------------------
-------------------------#

# For only one organism
   if  len(a_organism) ==1 and a_organism[0] != '':
      query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE
\"%{0}%\"'.format(a_organism[0])
      query = query + ');'
```

```
      q_h = q_h + ' Organism name(s) = ' + a_organism[0]

# for multiple organism
   elif len(a_organism) > 1 and a_organism[0] != '':
      q_h = q_h + ' Organism name(s) = '

      for i in range(0,len(a_organism)-1):
         query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE \"%{0}%\"
OR'.format(a_organism[i])
         q_h = q_h + a_organism[i] + ' or '
      query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE \"%{0}%\"
OR'.format(a_organism[-1])

      q_h = q_h +a_organism[-1]
      query = query + ');'


#-------------------------Length part-------------------------------
---------------------------#

# For organism query null
   if len(a_length) != 0 and (len(a_organism) == 0 or a_organism[0] ==
''):

      if length_range == 'less_than':
         query = query + 'PROTEINS.Length<' + a_length
         query = query + ');'
         q_h = q_h + '  Length < ' + a_length

      elif length_range == 'greater_than':
         query = query + 'PROTEINS.Length>' + a_length
         query = query + ');'
         q_h = q_h + '  Length > ' + a_length

      elif length_range == 'equal_to':

         query = query + 'PROTEINS.Length=' + a_length
         query = query + ');'
         q_h = q_h + '  Length = ' + a_length

#For organism query not null
   elif a_length != '' and (len(a_organism) >= 1 or a_organism[0] !=
''):
      query = query[:-1] + 'AND ('
      if length_range == 'less_than':
         query = query + 'PROTEINS.Length<' + a_length
         query = query + ');'
         q_h = q_h + ' + Length < ' + a_length

      elif length_range == 'greater_than':
```

```
         query = query + 'PROTEINS.Length>' + a_length
         query = query + ');'
         q_h = q_h + ' + Length > ' + a_length

      elif length_range == 'equal_to':
         query = query + 'PROTEINS.Length=' + a_length
         query = query + ');'

         q_h = q_h + ' + Length = ' + a_length



#------------------------GRAFS and IUPHAR ----------------------------
-----------------------------#

#Replacing the input to match the sql field names
   iuphar = [w.replace('ClassA','Class A') for w in iuphar]
   iuphar = [w.replace('ClassB','Class B') for w in iuphar]
   iuphar = [w.replace('ClassC','Class C') for w in iuphar]
   iuphar = [w.replace('ClassD','Class D') for w in iuphar]
   iuphar = [w.replace('ClassE','Class E') for w in iuphar]
   iuphar = [w.replace('ClassF','Class F') for w in iuphar]
   iuphar = [w.replace('ClassT2R','Class T2R') for w in iuphar]

   grafs = [w.replace('Glutamate','Glutamate-like') for w in grafs]
   grafs = [w.replace('Secretin','Secretin-like') for w in grafs]
   grafs = [w.replace('Rhodopsin','Rhodopsin-like') for w in grafs]
   grafs = [w.replace('Adhesion','Adhesion-like') for w in grafs]
   grafs = [w.replace('fungal','Fungal pheromone') for w in grafs]
   grafs = [w.replace('camp','cAMP receptor') for w in grafs]
   grafs = [w.replace('taste2r','Taste2R') for w in grafs]

# Query for organism and length both null for GRAFS and IUPHAR

   if a_length == '' and (len(a_organism) == 0 ) and ((len(grafs) != 0
) or (len(iuphar) != 0)):

#Query for GRAFS and IUPHAR both inputs
      if (len(grafs) != 0  ) and (len(iuphar) != 0):
         # Only one GRAFS input
         if len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
            q_h = q_h + ' GRAFS = ' + grafs[0]

         # Multiple GRAFS input
         elif  len(grafs) > 1:
            q_h = q_h + ' GRAFS = '

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
```

```
            q_h = q_h +  grafs[i] + ' or '
         query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])
         q_h = q_h  + grafs[-1]

      query = query + ' OR '
      q_h = q_h + ' or '

      # Only one IUPHAR input
      if  len(iuphar) == 1:
         query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
         query = query + ');'
         q_h = q_h + ' IUPHAR = ' + iuphar[0]

      #Multiple IUPHAR input
      elif  len(iuphar) > 1:
         q_h = q_h + ' IUPHAR = '

         for i in range(0,len(iuphar)-1):
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
            q_h = q_h + iuphar[i]+' or '

         query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])
         query = query + ');'
         q_h = q_h + iuphar[-1]

# If only GRAFS is used

   elif (len(grafs) != 0) and (len(iuphar) == 0 ):

      if len(grafs) == 1:
         query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
         query = query + ');'
         q_h = q_h + ' GRAFS = ' + grafs[0]

      elif  len(grafs) > 1:
         q_h = q_h + ' GRAFS = '

         for i in range(0,len(grafs)-1):
            query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
            q_h = q_h + grafs[i]+ ' or '

         query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])
         query = query + ');'

         q_h = q_h + grafs[-1]
```

```
# If only IUPHAR is used
      elif (len(iuphar) != 0 ) and (len(grafs) ==  0):

         if iuphar[0] != '' and len(iuphar) == 1:
            q_h = q_h + ' IUPHAR = ' + iuphar[0]

            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'
         elif  len(iuphar) > 1:
            q_h = q_h + ' IUPHAR = '

            for i in range(0,len(iuphar)-1):

               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
               q_h = q_h + iuphar[i] + ' or '

            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])
            query = query + ');'
            q_h = q_h + iuphar[-1]

# Length, orgnaism not null with GRAFS and IUPHAR
   elif (a_length != '' or (len(a_organism) != 0)) and ((len(grafs) !=
0) or (len(iuphar) != 0))  :

      query = query[:-1] + 'AND ('

      if (len(grafs) > 0 ) and (len(iuphar) > 0):

         if  len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
            q_h = q_h + ' + GRAFS = ' + grafs[0]

         elif  len(grafs) > 1:

            q_h = q_h + ' + GRAFS = '

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
               q_h = q_h + grafs[i]+ ' or '
            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])
            q_h = q_h + grafs[-1]


         query = query + ' OR '
         q_h = q_h + ' or '

         if  len(iuphar) == 1:
```

```
            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'
            q_h = q_h + ' IUPHAR = ' + iuphar[0]

         elif  len(iuphar) > 1:
            q_h = q_h + '  IUPHAR = '

            for i in range(0,len(iuphar)-1):
               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
               q_h = q_h + iuphar[i]+ ' or '
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])

            query = query + ');'
            q_h = q_h + iuphar[-1]


      elif (len(grafs) != 0 ) and (len(iuphar) == 0):
         query = query[:-1] + 'AND ('
         if  len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
            query = query + ');'
            q_h = q_h + ' + GRAFS = ' + grafs[0]

         elif  len(grafs) > 1:
            q_h = q_h + '+ GRAFS = '

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
               q_h = q_h + grafs[i] + ' or '
            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])

            query = query + ');'
            q_h = q_h  + grafs[-1]


      elif (len(iuphar) != 0) and (len(grafs) ==  0 ):
         query = query[:-1] + 'AND ('
         if  len(iuphar) == 1:
            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'
            q_h = q_h + ' + IUPHAR = ' + iuphar[0]

         elif  len(iuphar) > 1:
            q_h = q_h + ' + IUPHAR = '

            for i in range(0,len(iuphar)-1):
```

```
               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
               q_h = q_h +  iuphar[0]+ ' or '
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])

            query = query + ');'
            q_h = q_h + iuphar[-1]


#----------------------------Sub-family-------------------------------
----------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == ''):


      if  len(a_sub_fam) == 1:

         query = query + 'Ll_class.sub_family LIKE
\"%{0}%\"'.format(a_sub_fam[0])
         query = query + ');'
         q_h = q_h + ' Sub-family: ' + a_sub_fam[0]
      elif len(a_sub_fam) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Sub-family: '

         for i in range(0,len(a_sub_fam)-1):
            query = query + 'Ll_class.sub_family LIKE \"%{0}%\"
OR'.format(a_sub_fam[i])
            q_h = q_h + a_sub_fam[i] + ' or '

         query = query + ' Ll_class.sub_family LIKE \"%{0}\%"
'.format(a_sub_fam[-1])
         query = query + ');'
         q_h = q_h  + a_sub_fam[-1]


# When one or more queries are not null
   else:
      if  len(a_sub_fam) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.sub_family LIKE
\"%{0}%\"'.format(a_sub_fam[0])
         query = query + ');'
         q_h = q_h + ' + Sub-family: ' + a_sub_fam[0]

      elif len(a_sub_fam) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Sub-family: '
```

```
         for i in range(0,len(a_sub_fam)-1):
            query = query + '  Ll_class.sub_family LIKE \"%{0}%\"
OR'.format(a_sub_fam[i])
            q_h = q_h + a_sub_fam[i] + ' or '

         query = query + '  Ll_class.sub_family LIKE \"%{0}%\"
'.format(a_sub_fam[-1])
         query = query + ');'
         q_h = q_h  + a_sub_fam[-1]




#---------------------------Sub-sub-family--------------------------
-------------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') :

   #query = query[:-1] + 'AND ('
      print "dihofhof khane",a_sub_sub_fam
      if  len(a_sub_sub_fam) == 1:

         query = query + 'Ll_class.s_sub_family LIKE
\"%{0}%\"'.format(a_sub_sub_fam[0])
         query = query + ');'
         q_h = q_h + ' Sub-sub-family: ' + a_sub_sub_fam[0]
      elif len(a_sub_sub_fam) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Sub-sub-family: '

         for i in range(0,len(a_sub_sub_fam)-1):
            query = query + 'Ll_class.s_sub_family LIKE \"%{0}%\"
OR'.format(a_sub_sub_fam[i])
            q_h = q_h + a_sub_sub_fam[i] + ' or '

         query = query + ' Ll_class.sub_family LIKE \"%{0}\%"
'.format(a_sub_sub_fam[-1])
         query = query + ');'
         q_h = q_h  + a_sub_sub_fam[-1]


# When one or more queries are not null
   else:
      if  len(a_sub_sub_fam) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.s_sub_family LIKE
\"%{0}%\"'.format(a_sub_sub_fam[0])
         query = query + ');'
```

```
         q_h = q_h + ' + Sub-sub-family: ' + a_sub_sub_fam[0]

      elif len(a_sub_sub_fam) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Sub-sub-family: '

         for i in range(0,len(a_sub_sub_fam)-1):
            query = query + '  Ll_class.s_sub_family LIKE \"%{0}%\"
OR'.format(a_sub_sub_fam[i])
            q_h = q_h + a_sub_sub_fam[i] + ' or '

         query = query + '  Ll_class.s_sub_family LIKE \"%{0}%\"
'.format(a_sub_sub_fam[-1])
         query = query + ');'
         q_h = q_h  + a_sub_sub_fam[-1]



#---------------------------Sub-type---------------------------------
--------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='''
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == ''):

      #query = query[:-1] + 'AND ('

      #print "ami ekhane",a_sub_fam
      if  len(a_sub_type) == 1:

         query = query + 'Ll_class.sub_type LIKE
\"%{0}%\"'.format(a_sub_type[0])
         query = query + ');'
         q_h = q_h + ' Sub-type: ' + a_sub_type[0]
      elif len(a_sub_type) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Sub-type: '

         for i in range(0,len(a_sub_type)-1):
            query = query + 'Ll_class.sub_type LIKE \"%{0}%\"
OR'.format(a_sub_type[i])
            q_h = q_h + a_sub_fam[i] + ' or '

         query = query + ' Ll_class.sub_type LIKE \"%{0}\%"
'.format(a_sub_type[-1])
         query = query + ');'
         q_h = q_h  + a_sub_type[-1]

# When one or more queries are not null
```

```
   else:
      if  len(a_sub_type) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.sub_type LIKE
\"%{0}%\"'.format(a_sub_type[0])
         query = query + ');'
         q_h = q_h + ' + Sub-type: ' + a_sub_type[0]

      elif len(a_sub_type) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Sub-type: '

         for i in range(0,len(a_sub_type)-1):
            query = query + '  Ll_class.sub_type LIKE \"%{0}%\"
OR'.format(a_sub_type[i])
            q_h = q_h + a_sub_type[i] + ' or '

         query = query + '  Ll_class.sub_type LIKE \"%{0}%\"
'.format(a_sub_type[-1])
         query = query + ');'
         q_h = q_h  + a_sub_type[-1]

#-------------------------Protein name-------------------------------
----------------------------#

# For organism query null
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='''
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] != ''):
      if  len(a_pname) == 1:

         query = query + 'PROTEINS.Protein_Name LIKE
\"%{0}%\"'.format(a_pname[0])
         query = query + ');'
         q_h = q_h + ' Protein Name(s) = ' + a_pname[0]
      elif len(a_pname) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Protein Name(s) = '

         for i in range(0,len(a_pname)-1):
            query = query + 'PROTEINS.Protein_Name LIKE \"%{0}%\"
OR'.format(a_pname[i])
            q_h = q_h + a_pname[i] + ' or '

         query = query + ' PROTEINS.Protein_Name LIKE \"%{0}\%"
'.format(a_pname[-1])
         query = query + ');'
         q_h = q_h  + a_pname[-1]
```

```
# When one or more queries are not null
   else:
      if  len(a_pname) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.Protein_Name LIKE
\"%{0}%\"'.format(a_pname[0])
         query = query + ');'
         q_h = q_h + ' + Protein Name(s)= ' + a_pname[0]

      elif len(a_pname) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Protein Name(s)= '

         for i in range(0,len(a_pname)-1):
            query = query + '  PROTEINS.Protein_Name LIKE \"%{0}%\"
OR'.format(a_pname[i])
            q_h = q_h + a_pname[i] + ' or '

         query = query + '  PROTEINS.Protein_Name LIKE \"%{0}%\"
'.format(a_pname[-1])
         query = query + ');'
         q_h = q_h  + a_pname[-1]

#-------------------------Gene name----------------------------------
-------------------------#

# For organism query null
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] != '') and (len(a_pname) == 0
or a_pname[0] == '') :
      if  len(a_gname) == 1:

         query = query + 'PROTEINS.Gene_Name LIKE
\"%{0}%\"'.format(a_gname[0])
         query = query + ');'
         q_h = q_h + ' Gene Name(s) = ' + a_gname[0]
      elif len(a_gname) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Gene Name(s) =  '

         for i in range(0,len(a_gname)-1):
            query = query + 'PROTEINS.Gene_Name LIKE \"%{0}%\"
OR'.format(a_gname[i])
            q_h = q_h + a_gname[i] + ' or '

         query = query + ' PROTEINS.Gene_Name LIKE \"%{0}\%"
'.format(a_gname[-1])
         query = query + ');'
         q_h = q_h  + a_gname[-1]
```

```
# When one or more queries are not null
   else:
      if  len(a_gname) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.Gene_Name LIKE
\"%{0}%\"'.format(a_gname[0])
         query = query + ');'
         q_h = q_h + ' + Gene Name(s)= ' + a_gname[0]

      elif len(a_gname) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Gene Name(s)= '

         for i in range(0,len(a_gname)-1):
            query = query + '  PROTEINS.Gene_Name LIKE \"%{0}%\"
OR'.format(a_gname[i])
            q_h = q_h + a_gname[i] + ' or '

         query = query + '  PROTEINS.Gene_Name LIKE \"%{0}%\"
'.format(a_gname[-1])
         query = query + ');'
         q_h = q_h  + a_gname[-1]



#----------------------- AA part-------------------------------------
----------------------#

# When length, organism, GRAFS and IUPHAR are null except amino acid
percentages
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] == '') and (len(a_pname) == 0
or a_pname[0] == '') and (len(a_gname) == 0 or a_gname[0] == ''):


      if  len(aa) == 1:

         query = query + 'PROTEINS.{0}/100'.format(aa[0])
         query = query + ');'
         q_h = q_h + ' Amino acid percentage, ' + aa[0]
      elif len(aa) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Amino acid percentage, '

         for i in range(0,len(aa)-1):
            query = query + 'PROTEINS.{0}/100 OR'.format(aa[i])
```

```
            q_h = q_h + aa[i] + ' or '

         query = query + ' PROTEINS.{0}/100 '.format(aa[-1])
         query = query + ');'
         q_h = q_h  + aa[-1]

# When one or more queries are not null with amino acid percentages
   else:
      if  len(aa) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.{0}/100'.format(aa[0])
         query = query + ');'
         q_h = q_h + ' + Amino acid percentage, ' + aa[0]

      elif len(aa) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Amino acid percentage, '

         for i in range(0,len(aa)-1):
            query = query + '  PROTEINS.{0}/100 OR'.format(aa[i])
            q_h = q_h + aa[i] + ' or '

         query = query + '  PROTEINS.{0}/100 '.format(aa[-1])
         query = query + ');'
         q_h = q_h  + aa[-1]

#----------------------- Dipeptide part-------------------------------
----------------------------#

   dipep = [w.replace('AS', 'AS1') for w in dipep]
   dipep = [w.replace('IF', 'IF1') for w in dipep]
   dipep = [w.replace('IN', 'IN1') for w in dipep]
   dipep = [w.replace('IS', 'IS1') for w in dipep]


   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) == 0 or iuphar[0] == '') and (len(grafs) ==  0 or 
grafs[0] == '') and  (len(aa) ==  0 or aa[0] == '') and 
(len(a_sub_fam) == 0 or a_sub_fam[0] == '') and (len(a_sub_sub_fam) == 
0 or a_sub_sub_fam[0] == '') and (len(a_sub_type) == 0 or 
a_sub_type[0] == '') and (len(a_pname) == 0 or a_pname[0] == '') and 
(len(a_gname) == 0 or a_gname[0] == ''):
      if len(dipep) == 1:
         query = query + ' AA_Dipeptide.{0}'.format(dipep[0])
         query = query + ');'
         q_h = q_h + ' Dipeptide percentage, ' + dipep[0]

      elif len(dipep) > 1:
         query = query[:-1] + 'AND ('
         q_h = q_h + ' Dipeptide percentage, '

         for i in range(0,len(dipep)-1):
```

```
            query = query + '  AA_Dipeptide.{0} OR'.format(dipep[i])
            q_h = q_h + dipep[i] + ' or '
         query = query + '  AA_Dipeptide.{0} '.format(dipep[-1])
         query = query + ');'
         q_h = q_h + dipep[-1]

   else:
      if len(dipep) == 1:

         query = query[:-1] + 'AND('
         query = query + ' AA_Dipeptide.{0}'.format(dipep[0])
         query = query + ');'
         q_h = q_h + ' + Dipeptide percentage, ' + dipep[0]

      elif len(dipep) > 1:
         query = query[:-1] + 'AND('
         q_h = q_h + ' + Dipeptide percentage, '

         for i in range(0,len(dipep)-1):
            query = query + '  AA_Dipeptide.{0} OR'.format(dipep[i])
            q_h = q_h + dipep[i] + ' or '
         query = query + '  AA_Dipeptide.{0} '.format(dipep[-1])
         query = query + ');'
         q_h = q_h + dipep[-1]

# For removing query mistakes
   query = query.replace('AND AND', 'AND')
   query = query.replace('AND () AND', 'AND')
   query = query.replace(';) AND (',';')
   query = query.replace('WHERE AND', 'WHERE')
   query = query.replace('OR)',')')
   print query
#-------------------------------------------------------------------
------------------------------#

   cursor.execute(query)
   db.commit()

   numrows = int(cursor.rowcount)
   numrows1 = str(numrows)

# Creating Display (d_h) and Query headers (q_h)
   head_list = [numrows1]
   head_list.append(q_h)
   head_list.append(d_h)

   toReturn.insert(0,tuple(head_list))

   # get and display one row at a time.
   if numrows == 1:
      row = cursor.fetchone()
      toReturn.append(row)
```

```
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn.append(row)
   return toReturn

##########################################################################
########################################
#
#     For FASTA download
#
##########################################################################
########################################
#def fasta_download_db_quick(...):


cols_common = 'SELECT PROTEINS.PROTEIN_ID,PROTEINS.Sequence FROM
PROTEINS INNER JOIN Organism_v2 ON PROTEINS.Organism_ID =
Organism_v2.Organism_ID INNER JOIN GRAFS ON PROTEINS.GRAFS_ID =
GRAFS.GRAFS_ID INNER JOIN IUPHAR ON PROTEINS.IUPHAR_ID =
IUPHAR.IUPHAR_ID INNER JOIN AA_Dipeptide ON PROTEINS.PROTEIN_ID =
AA_Dipeptide.PROTEIN_ID INNER JOIN TMHMM_Length ON
TMHMM_Length.PROTEIN_ID = PROTEINS.PROTEIN_ID INNER JOIN Ll_class ON
PROTEINS.PROTEIN_ID = Ll_class.PROTEIN_ID'


def
a_search_fasta_db(a_organism,length_range,a_length,grafs,iuphar,a_sub_
fam,a_sub_sub_fam,a_sub_type,aa,dipep,display,pro_type,jobId):

   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


#=========================================================================
======
#
# Quick Search
#
#=========================================================================
======

#--------------Organism Search---------------------------------------
-------------------------#
def organism_search_fasta_db(q,jobId):
   import MySQLdb
```

```
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


   # execute SQL select statement

   query = '%s '%(cols_common) + 'WHERE Organism_v2.Common_name LIKE
\"%{0}%\" OR Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species
LIKE \"%{0}%\";'.format(q)

   print 'query organism'
   print query
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq

#----------------------------------------Entry Name----------------
-----------------------------------#
def entryname_search_fasta_db(q,jobId):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
```

```
   toReturn = []

   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE PROTEINS.Name LIKE
\"%{0}%\";'.format(q)
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq
#----------------------------------------Protein Name--------------
---------------------------------------#
def pname_search_fasta_db(q,jobId):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
   toReturn = []

   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE PROTEINS.Protein_Name LIKE
\"%{0}%\";'.format(q)
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
```

```
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq



#----------------------------------------Entry Name----------------
-----------------------------------#
def gname_search_fasta_db(q,jobId):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()
   toReturn = []

   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE PROTEINS.Gene_Name LIKE
\"%{0}%\";'.format(q)
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
```

```
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq


#-------------------------------------------------Uniprot ID--------
----------------------------------------#
def proid_search_fasta_db(q,jobId):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)


   cursor = db.cursor()


   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE PROTEINS.PROTEIN_ID LIKE
\"%{0}%\";'.format(q)

   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq

#---------------------------Classification-------------------------
---------------------------#
def classification_fasta_db(q,jobId):
```

```
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)


   cursor = db.cursor()
   # execute SQL select statement

#Query for non GPCRs
   if q == 'Non' or q == 'non' or q == 'None' or q == 'none':
      query = '%s '%(cols_common) + 'WHERE PROTEINS.PROTEIN_TYPE
="CN";'

   else:
#Query for GPCR classifications
      query = '%s '%(cols_common) + 'WHERE IUPHAR.IUPHAR LIKE
\"%{0}%\" OR GRAFS.GRAFS LIKE \"%{0}%\";'.format(q)
  # print query


   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []


   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq

#----------------------------AA percentage-----------------------------
--------------------------#
```

```
def aa_quick_fasta_db(q,jobId):
   import MySQLdb
   from Config import *

   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()

   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE
PROTEINS.\"{0}\/100";'.format(q)



   query = query.replace('"','')
   query = query.replace('\\','')

  # print query
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq


#----------------------------Length-----------------------------------
----------------------------#

def length_quick_fasta_db(q,jobId):
   import MySQLdb
   from Config import *
```

```
   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()


   # execute SQL select statement
   query = '%s '%(cols_common) + 'WHERE PROTEINS.Length
\"{0}\";'.format(q)

   query = query.replace('"','')
   query = query.replace('\\','')
   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
   fo.close()

   #print toReturn
   #return toReturn_seq


#=================================================================
======
#
# advanced SEARCH
#
#=================================================================
======
def
a_search_fasta_db(a_organism,a_pname,a_gname,length_range,a_length,gra
fs,iuphar,a_sub_fam,a_sub_sub_fam,a_sub_type,aa,dipep,display,pro_type
,jobId):

   import MySQLdb
   from Config import *
```

```
   # connect
   db = MySQLdb.connect(host=SQL_HOST, user=SQL_UN, passwd=SQL_PS,
   db=SQL_DB,read_default_file=SQL_CNF)

   cursor = db.cursor()

# If all the search term are null except "Display" and "Category"

   if (display != [] or pro_type != [])  and (len(a_organism) == 0 )
and a_length == '' and (len(iuphar) == 0 ) and (len(grafs) == 0 ) and
(len(a_sub_fam) ==0) and (len(a_sub_sub_fam) == 0) and
(len(a_sub_type) == 0) and (len(aa) == 0) and (len(dipep) == 0 ):
      if pro_type != [] or display != []:
         if len(pro_type) == 1:

            if pro_type[0] == 'cgf_type':
            #Every box is empty, only pro_type = cg_type
               query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF";'

            elif pro_type[0] == 'cg_type':
            #Every box is empty only pro_type = cn_type
             query = '%s '%(cols_common)+ ' WHERE PROTEINS.PROTEIN_TYPE
= "CG";'

            elif pro_type[0] == 'cn_type':
               query = '%s '%(cols_common) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN";'

            elif pro_type[0] == 'cnt_type':
               query = '%s '%(cols_common) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT";'

         elif len(pro_type) == 2:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type':

                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF";'


               elif pro_type[1] == 'cn_type':
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN";'

               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_common) + 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT";'

            elif pro_type[0] == 'cgf_type':
               if pro_type[1] == 'cn_type':
```

```
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN";'

               elif pro_type[1] == 'cnt_type':
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT";'

            else:
                query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT";'

         elif len(pro_type) == 3:
            if pro_type[0] == 'cg_type':
               if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN";'

               elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT";'

               elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT";'

            else:
                query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT";'

         else:
            query = '%s '%(cols_common)+ ';'




#-------------------when there is a query term-----------------------
-----------------------------------#

   elif (display != [] or pro_type != [])  and ((len(a_organism) != 0)
or (a_length != '') or (len(iuphar) != 0) or (len(grafs) != 0) or
(len(a_sub_fam) !=0) or (len(a_sub_sub_fam) != 0) or (len(a_sub_type)
!= 0) or (len(aa) != 0 ) or (len(dipep) != 0 )):
      if pro_type != [] or display != []:
```

```
        if len(pro_type) == 1:

           if pro_type[0] == 'cgf_type':
           #Every box is empty, only pro_type = cg_type
              query = '%s '%(cols_common)+ 'WHERE
PROTEINS.PROTEIN_TYPE = "CGF" AND ('

           elif pro_type[0] == 'cg_type':
           #Every box is empty only pro_type = cn_type

            query = '%s '%(cols_common)+ ' WHERE PROTEINS.PROTEIN_TYPE
= "CG" AND ('

           elif pro_type[0] == 'cn_type':
              query = '%s '%(cols_common) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CN" AND ('

           elif pro_type[0] == 'cnt_type':
                 query = '%s '%(cols_common) + ' WHERE
PROTEINS.PROTEIN_TYPE = "CNT" AND ('

        elif len(pro_type) == 2:
           if pro_type[0] == 'cg_type':
              if pro_type[1] == 'cgf_type':

                 query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CGF") AND ('

              elif pro_type[1] == 'cn_type':
                 query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN") AND ('

              elif pro_type[1] == 'cnt_type':
                 query = '%s '%(cols_common) + 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE = "CNT") AND ('

           elif pro_type[0] == 'cgf_type':
              if pro_type[1] == 'cn_type':
                 query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE = "CN") AND ('

              elif pro_type[1] == 'cnt_type':
                 query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('

           else:
              query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CN" OR PROTEINS.PROTEIN_TYPE ="CNT") AND ('

        elif len(pro_type) == 3:
           if pro_type[0] == 'cg_type':
```

```
               if pro_type[1] == 'cgf_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CN") AND ('

               elif pro_type[1] == 'cgf_type' and pro_type[2] ==
'cnt_type':
                  query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CGF" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('

               elif pro_type[1] == 'cnt_type' and pro_type[2] ==
'cn_type':
                  query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CG" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE ="CNT") AND ('

            else:
               query = '%s '%(cols_common)+ 'WHERE
(PROTEINS.PROTEIN_TYPE = "CGF" OR PROTEINS.PROTEIN_TYPE ="CN" OR
PROTEINS.PROTEIN_TYPE = "CNT") AND ('

         else:
            query = '%s '%(cols_common)+ 'WHERE ('


   else:

      query = '%s '%(cols_common) + 'WHERE ('

      #toReturn = [('ID','Entry
name','Length','Type','Genus','Species','Common
name','GRAFS','IUPHAR')]



#----------------------------Organism part-----------------------------
-------------------------#

# For only one organism
   if  len(a_organism) ==1 and a_organism[0] != '':
      query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE
\"%{0}%\"'.format(a_organism[0])
      query = query + ');'

# for multiple organism
   elif len(a_organism) > 1 and a_organism[0] != '':

      for i in range(0,len(a_organism)-1):
```

```
         query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE \"%{0}%\"
OR'.format(a_organism[i])
      query = query + ' Organism_v2.Common_name LIKE \"%{0}%\" OR
Organism_v2.Genus LIKE \"%{0}%\" OR Organism_v2.Species LIKE \"%{0}%\"
OR'.format(a_organism[-1])

      query = query + ');'


#-------------------------Length part-------------------------------
---------------------------#

# For organism query null
   if a_length != '' and (len(a_organism) == 0 or a_organism[0] ==
''):

      if length_range == 'less_than':
         query = query + 'PROTEINS.Length<' + a_length
         query = query + ');'

      elif length_range == 'greater_than':
         query = query + 'PROTEINS.Length>' + a_length
         query = query + ');'

      elif length_range == 'equal_to':

         query = query + 'PROTEINS.Length=' + a_length
         query = query + ');'

#For organism query not null
   elif a_length != '' and (len(a_organism) >= 1 or a_organism[0] !=
''):
      query = query[:-1] + 'AND ('
      if length_range == 'less_than':
         query = query + 'PROTEINS.Length<' + a_length
         query = query + ');'

      elif length_range == 'greater_than':
         query = query + 'PROTEINS.Length>' + a_length
         query = query + ');'

      elif length_range == 'equal_to':
         query = query + 'PROTEINS.Length=' + a_length
         query = query + ');'


#-----------------------GRAFS and IUPHAR ---------------------------
---------------------------#

#Replacing the input to match the sql field names
   iuphar = [w.replace('ClassA','Class A') for w in iuphar]
```

```
   iuphar = [w.replace('ClassB','Class B') for w in iuphar]
   iuphar = [w.replace('ClassC','Class C') for w in iuphar]
   iuphar = [w.replace('ClassD','Class D') for w in iuphar]
   iuphar = [w.replace('ClassE','Class E') for w in iuphar]
   iuphar = [w.replace('ClassF','Class F') for w in iuphar]
   iuphar = [w.replace('ClassT2R','Class T2R') for w in iuphar]

   grafs = [w.replace('Glutamate','Glutamate-like') for w in grafs]
   grafs = [w.replace('Secretin','Secretin-like') for w in grafs]
   grafs = [w.replace('Rhodopsin','Rhodopsin-like') for w in grafs]
   grafs = [w.replace('Adhesion','Adhesion-like') for w in grafs]
   grafs = [w.replace('fungal','Fungal pheromone') for w in grafs]
   grafs = [w.replace('camp','cAMP receptor') for w in grafs]
   grafs = [w.replace('taste2r','Taste2R') for w in grafs]
# Query for organism and length both null for GRAFS and IUPHAR

   if a_length == '' and (len(a_organism) == 0 ) and ((len(grafs) != 0
) or (len(iuphar) != 0)):

#Query for GRAFS and IUPHAR both inputs
      if (len(grafs) != 0  ) and (len(iuphar) != 0):
         # Only one GRAFS input
         if len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])

         # Multiple GRAFS input
         elif  len(grafs) > 1:

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])
         query = query + ' OR '

         # Only one IUPHAR input
         if  len(iuphar) == 1:
            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'

         #Multiple IUPHAR input
         elif  len(iuphar) > 1:

            for i in range(0,len(iuphar)-1):
               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])

            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])
            query = query + ');'
```

```
# If only GRAFS is used

      elif (len(grafs) != 0) and (len(iuphar) == 0 ):

         if len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
            query = query + ');'

         elif  len(grafs) > 1:

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])

            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])
            query = query + ');'

# If only IUPHAR is used
      elif (len(iuphar) != 0 ) and (len(grafs) ==  0):

         if iuphar[0] != '' and len(iuphar) == 1:

            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'

         elif  len(iuphar) > 1:

            for i in range(0,len(iuphar)-1):

               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])
            query = query + ');'

# Length, orgnaism not null with GRAFS and IUPHAR
   elif (a_length != '' or (len(a_organism) != 0)) and ((len(grafs) !=
0) or (len(iuphar) != 0))  :

      query = query[:-1] + 'AND ('

      if (len(grafs) > 0 ) and (len(iuphar) > 0):

         if  len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])

         elif  len(grafs) > 1:

            for i in range(0,len(grafs)-1):
```

```
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])

         query = query + ' OR '

         if  len(iuphar) == 1:
            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'

         elif  len(iuphar) > 1:

            for i in range(0,len(iuphar)-1):
               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])

            query = query + ');'

      elif (len(grafs) != 0 ) and (len(iuphar) == 0):
         query = query[:-1] + 'AND ('
         if  len(grafs) == 1:
            query = query + ' GRAFS.GRAFS = \"{0}\"'.format(grafs[0])
            query = query + ');'

         elif  len(grafs) > 1:

            for i in range(0,len(grafs)-1):
               query = query + ' GRAFS.GRAFS = \"{0}\"
OR'.format(grafs[i])
            query = query + ' GRAFS.GRAFS = \"{0}\" '.format(grafs[-
1])

            query = query + ');'

      elif (len(iuphar) != 0) and (len(grafs) ==  0 ):
         query = query[:-1] + 'AND ('
         if  len(iuphar) == 1:
            query = query + ' IUPHAR.IUPHAR =
\"{0}\"'.format(iuphar[0])
            query = query + ');'

         elif  len(iuphar) > 1:

            for i in range(0,len(iuphar)-1):
               query = query + ' IUPHAR.IUPHAR = \"{0}\"
OR'.format(iuphar[i])
            query = query + ' IUPHAR.IUPHAR = \"{0}\"
'.format(iuphar[-1])
```

```
            query = query + ');'


#----------------------------Sub-family-------------------------------
----------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or 
grafs[0] == ''):

   #query = query[:-1] + 'AND ('

      print "ami ekhane",a_sub_fam
      if  len(a_sub_fam) == 1:

         query = query + 'Ll_class.sub_family LIKE 
\"%{0}%\"'.format(a_sub_fam[0])
         query = query + ');'

      elif len(a_sub_fam) > 1:
         query = query[:-1] + 'AND ('


         for i in range(0,len(a_sub_fam)-1):
            query = query + 'Ll_class.sub_family LIKE \"%{0}%\" 
OR'.format(a_sub_fam[i])


         query = query + ' Ll_class.sub_family LIKE \"%{0}\%" 
'.format(a_sub_fam[-1])
         query = query + ');'


# When one or more queries are not null
   else:
      if  len(a_sub_fam) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.sub_family LIKE 
\"%{0}%\"'.format(a_sub_fam[0])
         query = query + ');'


      elif len(a_sub_fam) > 1:
         query = query[:-1] + 'AND('


         for i in range(0,len(a_sub_fam)-1):
            query = query + '  Ll_class.sub_family LIKE \"%{0}%\" 
OR'.format(a_sub_fam[i])
```

```
         query = query + '  Ll_class.sub_family LIKE \"%{0}%\"
'.format(a_sub_fam[-1])
         query = query + ');'




#----------------------------Sub-sub-family----------------------------
--------------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') :


      if  len(a_sub_sub_fam) == 1:

         query = query + 'Ll_class.s_sub_family LIKE
\"%{0}%\"'.format(a_sub_sub_fam[0])
         query = query + ');'

      elif len(a_sub_sub_fam) > 1:
         query = query[:-1] + 'AND ('


         for i in range(0,len(a_sub_sub_fam)-1):
            query = query + 'Ll_class.s_sub_family LIKE \"%{0}%\"
OR'.format(a_sub_sub_fam[i])


         query = query + ' Ll_class.sub_family LIKE \"%{0}\%"
'.format(a_sub_sub_fam[-1])
         query = query + ');'



# When one or more queries are not null
   else:
      if  len(a_sub_sub_fam) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.s_sub_family LIKE
\"%{0}%\"'.format(a_sub_sub_fam[0])
         query = query + ');'


      elif len(a_sub_sub_fam) > 1:
         query = query[:-1] + 'AND('
```

```
         for i in range(0,len(a_sub_sub_fam)-1):
            query = query + '  Ll_class.s_sub_family LIKE \"%{0}%\"
OR'.format(a_sub_sub_fam[i])


         query = query + '  Ll_class.s_sub_family LIKE \"%{0}%\"
'.format(a_sub_sub_fam[-1])
         query = query + ');'




#----------------------------Sub-type---------------------------------
--------------------#

# When length, organism, GRAFS and IUPHAR are null except Sub-family
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length ==''
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == ''):

      #query = query[:-1] + 'AND ('

      #print "ami ekhane",a_sub_fam
      if  len(a_sub_type) == 1:

         query = query + 'Ll_class.sub_type LIKE
\"%{0}%\"'.format(a_sub_type[0])

      elif len(a_sub_type) > 1:
         query = query[:-1] + 'AND ('


         for i in range(0,len(a_sub_type)-1):
            query = query + 'Ll_class.sub_type LIKE \"%{0}%\"
OR'.format(a_sub_type[i])


         query = query + ' Ll_class.sub_type LIKE \"%{0}\%"
'.format(a_sub_type[-1])
         query = query + ');'

# When one or more queries are not null
   else:
      if  len(a_sub_type) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' Ll_class.sub_type LIKE
\"%{0}%\"'.format(a_sub_type[0])
         query = query + ');'
```

```
      elif len(a_sub_type) > 1:
         query = query[:-1] + 'AND('


         for i in range(0,len(a_sub_type)-1):
            query = query + '  Ll_class.sub_type LIKE \"%{0}%\"
OR'.format(a_sub_type[i])

         query = query + '  Ll_class.sub_type LIKE \"%{0}%\"
'.format(a_sub_type[-1])
         query = query + ');'


#-------------------------Protein name-------------------------------
----------------------------#

# For organism query null
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length ==''
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] != ''):
      if  len(a_pname) == 1:

         query = query + 'PROTEINS.Protein_Name LIKE
\"%{0}%\"'.format(a_pname[0])
         query = query + ');'

      elif len(a_pname) > 1:
         query = query[:-1] + 'AND ('


         for i in range(0,len(a_pname)-1):
            query = query + 'PROTEINS.Protein_Name LIKE \"%{0}%\"
OR'.format(a_pname[i])


         query = query + ' PROTEINS.Protein_Name LIKE \"%{0}\%"
'.format(a_pname[-1])
         query = query + ');'


# When one or more queries are not null
   else:
      if  len(a_pname) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.Protein_Name LIKE
\"%{0}%\"'.format(a_pname[0])
         query = query + ');'


      elif len(a_pname) > 1:
```

```
         query = query[:-1] + 'AND('


         for i in range(0,len(a_pname)-1):
            query = query + '  PROTEINS.Protein_Name LIKE \"%{0}%\"
OR'.format(a_pname[i])


         query = query + '  PROTEINS.Protein_Name LIKE \"%{0}%\"
'.format(a_pname[-1])
         query = query + ');'


#-------------------------Gene name----------------------------------
-------------------------#

# For organism query null
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] != '') and (len(a_pname) == 0
or a_pname[0] == '') :
      if  len(a_gname) == 1:

         query = query + 'PROTEINS.Gene_Name LIKE
\"%{0}%\"'.format(a_gname[0])
         query = query + ');'

      elif len(a_gname) > 1:
         query = query[:-1] + 'AND ('


         for i in range(0,len(a_gname)-1):
            query = query + 'PROTEINS.Gene_Name LIKE \"%{0}%\"
OR'.format(a_gname[i])


         query = query + ' PROTEINS.Gene_Name LIKE \"%{0}\%"
'.format(a_gname[-1])
         query = query + ');'



# When one or more queries are not null
   else:
      if  len(a_gname) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.Gene_Name LIKE
\"%{0}%\"'.format(a_gname[0])
         query = query + ');'
```

```
      elif len(a_gname) > 1:
         query = query[:-1] + 'AND('

         for i in range(0,len(a_gname)-1):
            query = query + '  PROTEINS.Gene_Name LIKE \"%{0}%\"
OR'.format(a_gname[i])


         query = query + '  PROTEINS.Gene_Name LIKE \"%{0}%\"
'.format(a_gname[-1])
         query = query + ');'




#----------------------- AA part-------------------------------------
----------------------#

# When length, organism, GRAFS and IUPHAR are null except amino acid
percentages
   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='
and (len(iuphar) ==  0 or iuphar[0] == '') and (len(grafs) ==  0 or
grafs[0] == '') and (len(a_sub_fam) == 0 or a_sub_fam[0] == '') and
(len(a_sub_sub_fam) == 0 or a_sub_sub_fam[0] == '') and
(len(a_sub_type) == 0 or a_sub_type[0] == '') and (len(a_pname) == 0
or a_pname[0] == '') and (len(a_gname) == 0 or a_gname[0] == '') :


      if  len(aa) == 1:

         query = query + 'PROTEINS.{0}/100'.format(aa[0])
         query = query + ');'
      elif len(aa) > 1:
         query = query[:-1] + 'AND ('

         for i in range(0,len(aa)-1):
            query = query + 'PROTEINS.{0}/100 OR'.format(aa[i])

         query = query + ' PROTEINS.{0}/100 '.format(aa[-1])
         query = query + ');'

# When one or more queries are not null with amino acid percentages
   else:
      if  len(aa) == 1:
         query = query[:-1] + 'AND ('
         query = query + ' PROTEINS.{0}/100'.format(aa[0])
         query = query + ');'

      elif len(aa) > 1:
         query = query[:-1] + 'AND('
```

```
         for i in range(0,len(aa)-1):
            query = query + '  PROTEINS.{0}/100 OR'.format(aa[i])

         query = query + '  PROTEINS.{0}/100 '.format(aa[-1])
         query = query + ');'

#----------------------- Dipeptide part-----------------------------
----------------------------#

   dipep = [w.replace('AS', 'AS1') for w in dipep]
   dipep = [w.replace('IF', 'IF1') for w in dipep]
   dipep = [w.replace('IN', 'IN1') for w in dipep]
   dipep = [w.replace('IS', 'IS1') for w in dipep]


   if (len(a_organism) == 0 or a_organism[0] == '') and a_length =='' 
and (len(iuphar) == 0 or iuphar[0] == '') and (len(grafs) ==  0 or 
grafs[0] == '') and  (len(aa) ==  0 or aa[0] == '') and 
(len(a_sub_fam) == 0 or a_sub_fam[0] == '') and (len(a_sub_sub_fam) == 
0 or a_sub_sub_fam[0] == '') and (len(a_sub_type) == 0 or 
a_sub_type[0] == '') and (len(a_pname) == 0 or a_pname[0] == '') and 
(len(a_gname) == 0 or a_gname[0] == '') :
      if len(dipep) == 1:
         query = query + ' AA_Dipeptide.{0}'.format(dipep[0])
         query = query + ');'

      elif len(dipep) > 1:
         query = query[:-1] + 'AND ('

         for i in range(0,len(dipep)-1):
            query = query + '  AA_Dipeptide.{0} OR'.format(dipep[i])
         query = query + '  AA_Dipeptide.{0} '.format(dipep[-1])
         query = query + ');'

   else:
      if len(dipep) == 1:

         query = query[:-1] + 'AND('
         query = query + ' AA_Dipeptide.{0}'.format(dipep[0])
         query = query + ');'

      elif len(dipep) > 1:
         query = query[:-1] + 'AND('

         for i in range(0,len(dipep)-1):
            query = query + '  AA_Dipeptide.{0} OR'.format(dipep[i])
         query = query + '  AA_Dipeptide.{0} '.format(dipep[-1])
         query = query + ');'

# For removing query mistakes
   query = query.replace('AND AND', 'AND')
```

```
   query = query.replace('AND () AND', 'AND')
   query = query.replace(';) AND (',';')
   query = query.replace('WHERE AND', 'WHERE')
   query = query.replace('OR)',')')
   print 'query print'
   print query
#-------------------------------------------------------------------
------------------------------#
#Creating fasta file

   cursor.execute(query)
   db.commit()
   numrows = int(cursor.rowcount)
   toReturn_seq = []
   if numrows == 1:
      row = cursor.fetchone()
      toReturn_seq.append(row)
   else:
      for x in range(0,numrows):
         row = cursor.fetchone()
         toReturn_seq.append(row)

   fo = open('%s%s_query_seq.fasta'%(OUTPUT_DIR,jobId),'w')
   for head in toReturn_seq:
      fo.write('>')
      fo.write(head[0])
      fo.write('\n')
      fo.write(head[1])
      fo.write('\n')
      #print head
   fo.close()

   return toReturn_seq
```

APPENDIX K: WEB-SERVER OUTPUT PAGE

```
$def with (setup,rows,link,fastalink)
<html>
<head>
     <meta charset="utf-8" />
     <title>UTEP GPCR Results </title>
     $:{setup['css']}
     $:{setup['favIco']}
    <!--   <script type="text/javascript"
src="../static/js/Table_Sorter_and_Filter.js"></script>  -->
               <script
src="../UTEP_GPCR/static/js/jquery.js"></script>
          <script src="../UTEP_GPCR/static/js/sorttable.js"></script>
          <script src="../UTEP_GPCR/static/js/filter.js"></script>
```

```
          <script
src="../UTEP_GPCR/static/js/floatThead/jquery.floatThead.min.js"></scr
ipt>
          <script
src="../UTEP_GPCR/static/js/expandContract.js"></script>


        <script>
           jQuery(window).load(function () {
                console.log('page is loaded');
                // May need to put modal here and close it when the
page finished loading
                jQuery(window).scroll(function() {
                     var table = jQuery('#table');
                     table.floatThead(); // Float the table
headersheaders
                });
           });


        </script>

</head>
<body>
     <div id="page-wrap">
     <div id="header-wrap">     $:{setup['banner']} </div>

          <hr>
          $:{setup['navBar']}
     <br/>
     <div id="main-content">



<h2>Results Table</h2>

<br>




     <h6>   <b> The number of proteins found: </b> $:rows[0][0] </h5>

     <h6>    <b>Query terms: </b>  $:rows[0][1]</h6>

<h6>    <b>Display: </b>  $:rows[0][2]</h6>

    <b>Download:</b> <a href="${link}" download target='_blank'>Result
table(csv) </a>


    <a href="${fastalink}">FASTA file</a>
```

```
<br>
<br>

  <input type="text" placeholder="Search" id="search"
class="search_box" style="width:25%; margin-left:25"
onkeyup="filter(this.id)">

        <div class="row">
          <table border="1" id="table" name="table" class="sortable" >
           <tr style="cursor:hand;">



                $for hdr in rows [1][0:]:
                     <th>${hdr}</th>
           </tr>
                $if len(rows) > 2:
                $for row in rows[2:]:
                     <tr><td><a
href="https://www.uniprot.org/uniprot/$:{row[0]}"
target='_blank'>$:{row[0]}</td>
                     $for item in row[1:-1]:
                          $if len(str(item)) > 40 and "href=" not in
item:
                               <td>
                               <div __state="contracted"
onclick="toggleExpand(this);">
                               <div class="expanded"
style="display:none">$:{item} </div>
                               <div class="contracted"
style="display:block"> $:{str(item)[:40]} ...</div>
                               </div>
                               </td>

                          $else:
                                <td>$:{item}</td>
                     <td>
                     $if len(row[-1]) > 50:
                          <div __state="contracted"
onclick="toggleExpand(this);">
                          <div class="contracted"
style="display:block">
                          $for pdb_row in row[-1].split('  ')[0:10]:
                               <a
href="https://www.rcsb.org/structure/$:{pdb_row}"
target=\'_blank\'>$:{pdb_row} </a><br>
                          ...</div>
                          <div class="expanded" style="display:none">
```

```
                              $for pdb_row in row[-1].split('  '):
                                   <a
href="https://www.rcsb.org/structure/$:{pdb_row}"
target=\'_blank\'>$:{pdb_row} </a><br>
                              </div>
                              </div>

                         $elif len(row[-1]) > 1:
                              $for pdb in row[-1].split('  '):
                                   <a
href="https://www.rcsb.org/structure/$:{pdb}" target='_blank'>$:{pdb}
</a><br>
                         $else:
                              $:{row[-1]}
                         </td></tr>


                    </table>
                   Click <a href="../kbegum/database"> here </a> to
return to the main search page.

              $else:
                   </table> Click <a href="../kbegum/database"> here </a>
to return to the main search page.




         </div>
        </div> <!-- main-content -->

$:{setup['menu']}
</body>
</html>
```

APPENDIX L: PARSING RESULT FROM SVM-PROT AND GPCR-CA

SVM-PROT

```
import numpy
import re

infile = open("3063_gpcr.xls","r")
#infile = open("full_length_1_try.xls","r")

lines = infile.read()

#print lines
```

```
#The index of the word Query repeated in the .xls file
start_position1 = [m.start() for m in re.finditer('Query=',lines)]

query = ''

for start in start_position1:
     x = lines[start:]
     stop = x.find('//')
     y = x[:stop]
     query = query + y
#    print query


fo = open('ID_length_3063_gpcr.csv','a')
#fo = open('ID_length.csv','a')


words1 = query.split()

m = 0
n = 0
for word in words1:

     n = n + 1
     #print word
     if word.startswith('Query='):
          m = m + 1

          ID = word.replace('Query=','')
          fo.write(str(m))
          fo.write(',')
          fo.write(ID)
          fo.write(',')
          print str(m)

     if word.startswith('Length='):
          L = word.replace('Length=','')
          fo.write(L)
          fo.write('\n')

fo.close()

# For the molecular function
start_position2 = [m.start() for m in re.finditer('Your protein may
belong to the following families:',lines)]

# The total number of authors
all_mol_type = ''
sep_mol_type = []

for start in start_position2:
```

```
      x = lines[start:]
      stop = x.find('//')
      y = x[:stop]
      all_mol_type = y

#     print all_mol_type

      words2 = all_mol_type.split('\r\n')
      sep_mol_type.append(words2)

#print sep_mol_type



fo1 = open("type_3063_gpcr.csv","a")
#fo1 = open("type.csv","a")

#Cleaning the list from \t\t or ''
for seq_list in sep_mol_type:
      for seq in seq_list:

            while seq == '\t\t' in seq_list:
                  seq_list.remove(seq)

            while seq == '' in seq_list:
                  seq_list.remove(seq)


#print sep_mol_type

#Checking the length
#for seq_list in sep_mol_type:
      #print len(seq_list)
      #print seq_list

filtered_list_mol_func = [] #saves only molecular function

filtered_list_bio_proc = [] #saves only biological procession

filtered_list_broad_def_func = [] #saves only broadly defined function

from itertools import groupby


for seq_list in sep_mol_type:

      #Only Molecular function and Biological process is present
      if ('Molecular Function:' in seq_list) and ('Biological Process:'
in seq_list) and ('Broadly Defined Function:' not in seq_list):
            #print seq_list
            filtered_list = [list(group) for k, group in
groupby(seq_list, lambda x: x == "Biological Process:") if not k]
```

```
            filtered_list_mol_func = filtered_list[0]
            filtered_list_bio_proc = filtered_list[1]


            if len(filtered_list_mol_func) == 4:
                  #print 'only 1'
                  a = filtered_list_mol_func[3].split('\t')
                  a[2] = float(a[2])

                  if (a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90:

                        #print 'PG'
                        #fo1.write('PG\n')
                        if (a[0].startswith('7')):
                              fo1.write(str(a[0]))
                              fo1.write('\n')
                        else:
                              fo1.write('PG\n')
                  else:

                        #print 'PN'
                        fo1.write('PN\n')


            elif len(filtered_list_mol_func) == 5:
                  #print 'only 2'
                  a = filtered_list_mol_func[3].split('\t')
                  b = filtered_list_mol_func[4].split('\t')

                  a[2] = float(a[2])
                  b[2] = float(b[2])

                  if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90):

                        #print 'PG'
                        #fo1.write('PG\n')

                        if(a[0].startswith('7')):
                              fo1.write(str(a[0]))
                              fo1.write('\n')

                        else:
                              fo1.write('PG\n')
                  else:

                        #print 'PN'
                        fo1.write('PN\n')
```

```
          elif len(filtered_list_mol_func) > 5:
               #print '3 or more'
               a = filtered_list_mol_func[3].split('\t')
               b = filtered_list_mol_func[4].split('\t')
               c = filtered_list_mol_func[5].split('\t')

               a[2] = float(a[2])
               b[2] = float(b[2])
               c[2] = float(c[2])


               if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90) or ((c[0].startswith('G') or c[0].startswith('7')) and c[2] >
90):

                    #fo1.write('PG\n')
                    if(a[0].startswith('7')):
                         fo1.write(str(a[0]))
                         fo1.write('\n')

                    elif (b[0].startswith('7')):
                         fo1.write(str(b[0]))
                         fo1.write('\n')

                    else:
                         fo1.write('PG\n')
               else:

                    #print 'PN'
                    fo1.write('PN\n')



     elif ('Molecular Function:' in seq_list) and  ('Broadly Defined
Function:' in seq_list) and ('Biological Process:' not in seq_list):

          filtered_list = [list(group) for k, group in
groupby(seq_list, lambda x: x == "Broadly Defined Function:") if not
k]

          filtered_list_mol_func = filtered_list[0]
          filtered_list_broad_def_func = filtered_list[1]

          #filtered_list = filtered_list[0]


          if len(filtered_list_mol_func) == 4:
               #print 'only 1'
               a = filtered_list_mol_func[3].split('\t')
```

```
                a[2] = float(a[2])

                if (a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90:

                     #print 'PG'
#                    fo1.write('PG\n')
                     if (a[0].startswith('7')):
                          fo1.write(str(a[0]))
                          fo1.write('\n')

                     else:
                          fo1.write('PG\n')
                else:

                     if len(filtered_list_broad_def_func) == 1:
                          x =
filtered_list_broad_def_func[0].split('\t')
                          x[2] = float(x[2])

                          if x[0].startswith('Transmembrane') and
x[2] > 90:

                               #print 'PNT'
                               fo1.write('PNT\n')

                          else:

                               #print 'PN'
                               fo1.write('PN\n')

                     elif len(filtered_list_broad_def_func) > 1:

                          x =
filtered_list_broad_def_func[0].split('\t')
                          y =
filtered_list_broad_def_func[1].split('\t')

                          x[2] = float(x[2])
                          y[2] = float(y[2])

                          if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):

                               #print 'PNT'
                               fo1.write('PNT\n')

                          else:

                               #print 'PN'
                               fo1.write('PN\n')
```

```
            elif len(filtered_list_mol_func) == 5:
                  #print 'only 2'
                  a = filtered_list_mol_func[3].split('\t')
                  b = filtered_list_mol_func[4].split('\t')

                  a[2] = float(a[2])
                  b[2] = float(b[2])

                  if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90):

                        #print 'PG'
                        #fo1.write('PG\n')
                        if (a[0].startswith('7')):
                              fo1.write(str(a[0]))
                              fo1.write('\n')


                        elif (b[0].startswith('7')):
                              fo1.write(str(b[0]))
                              fo1.write('\n')

                        else:
                              fo1.write('PG\n')
                  else:
                        if len(filtered_list_broad_def_func) == 1:
                              x =
filtered_list_broad_def_func[0].split('\t')
                              x[2] = float(x[2])

                              if x[0].startswith('Transmembrane') and
x[2] > 90:

                                    #print 'PNT'
                                    fo1.write('PNT\n')

                              else:

                                    #print 'PN'
                                    fo1.write('PN\n')

                        elif len(filtered_list_broad_def_func) > 1:

                              x =
filtered_list_broad_def_func[0].split('\t')
                              y =
filtered_list_broad_def_func[1].split('\t')

                              x[2] = float(x[2])
                              y[2] = float(y[2])
```

```
                        if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):

                              #print 'PNT'
                              fo1.write('PNT\n')

                        else:

                              #print 'PN'
                              fo1.write('PN\n')

          elif len(filtered_list_mol_func) > 5:
               #print '3 or more'
               a = filtered_list_mol_func[3].split('\t')
               b = filtered_list_mol_func[4].split('\t')
               c = filtered_list_mol_func[5].split('\t')

               a[2] = float(a[2])
               b[2] = float(b[2])
               c[2] = float(c[2])


               if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90) or ((c[0].startswith('G') or c[0].startswith('7')) and c[2] >
90):

                    #print 'PG'
               #    fo1.write('PG\n')

                    if (a[0].startswith('7')):
                         fo1.write(str(a[0]))
                         fo1.write('\n')

                    elif (b[0].startswith('7')):
                         fo1.write(str(b[0]))
                         fo1.write('\n')

                    elif (c[0].startswith('7')):
                         fo1.write(str(c[0]))
                         fo1.write('\n')

                    else:
                         fo1.write('PG\n')



               else:

                    if len(filtered_list_broad_def_func) == 1:
```

```
                              x =
filtered_list_broad_def_func[0].split('\t')
                              x[2] = float(x[2])

                              if x[0].startswith('Transmembrane') and
x[2] > 90:

                                    #print 'PNT'
                                    fo1.write('PNT\n')

                              else:

                                    #print 'PN'
                                    fo1.write('PN\n')

                        elif len(filtered_list_broad_def_func) > 1:

                              x =
filtered_list_broad_def_func[0].split('\t')
                              y =
filtered_list_broad_def_func[1].split('\t')

                              x[2] = float(x[2])
                              y[2] = float(y[2])

                              if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):

                                    #print 'PNT'
                                    fo1.write('PNT\n')

                              else:

                                    #print 'PN'
                                    fo1.write('PN\n')


      # All present (Molecular, Biology + Broadly)
      elif ('Molecular Function:' in seq_list) and  ('Broadly Defined
Function:' in seq_list) and ('Biological Process:' in seq_list):
            #print seq_list
            filtered_list = [list(group) for k, group in
groupby(seq_list, lambda x: x == "Biological Process:") if not k]

            filtered_list1 = [list(group) for k, group in
groupby(filtered_list[1], lambda x: x == "Broadly Defined Function:")
if not k]

            #print filtered_list[0]

            filtered_list_mol_func = filtered_list[0]
            filtered_list_bio_proc = filtered_list1[0]
```

```
            filtered_list_broad_def_func = filtered_list1[1]


            if len(filtered_list_mol_func) == 4:
                  #print 'only 1'
                  a = filtered_list_mol_func[3].split('\t')
                  a[2] = float(a[2])

                  if (a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90:

                        #print 'PG'
                        #fo1.write('PG\n')

                        if (a[0].startswith('7')):
                              fo1.write(str(a[0]))
                              fo1.write('\n')

                        else:
                              fo1.write('PG\n')

                  else:

                        if len(filtered_list_broad_def_func) == 1:
                              x =
filtered_list_broad_def_func[0].split('\t')
                              x[2] = float(x[2])

                              if x[0].startswith('Transmembrane') and
x[2] > 90:

                                    #print 'PNT'
                                    fo1.write('PNT\n')

                              else:

                                    #print 'PN'
                                    fo1.write('PN\n')

                        elif len(filtered_list_broad_def_func) > 1:

                              x =
filtered_list_broad_def_func[0].split('\t')
                              y =
filtered_list_broad_def_func[1].split('\t')

                              x[2] = float(x[2])
                              y[2] = float(y[2])

                              if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):
```

```
                              #print 'PNT'
                              fo1.write('PNT\n')

                         else:

                              #print 'PN'
                              fo1.write('PN\n')


          elif len(filtered_list_mol_func) == 5:
               #print 'only 2'
               a = filtered_list_mol_func[3].split('\t')
               b = filtered_list_mol_func[4].split('\t')

               a[2] = float(a[2])
               b[2] = float(b[2])

               if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90):

                    #print 'PG'
                    #fo1.write('PG\n')
                    if (a[0].startswith('7')):
                         fo1.write(str(a[0]))

                         fo1.write('\n')

                    elif (b[0].startswith('7')):
                         fo1.write(str(b[0]))
                         fo1.write('\n')


                    else:


                         fo1.write('PG\n')
               else:
                    if len(filtered_list_broad_def_func) == 1:
                         x =
filtered_list_broad_def_func[0].split('\t')
                         x[2] = float(x[2])

                         if x[0].startswith('Transmembrane') and
x[2] > 90:

                              #print 'PNT'
                              fo1.write('PNT\n')

                         else:

                              #print 'PN'
```

```
                              fo1.write('PN\n')

                    elif len(filtered_list_broad_def_func) > 1:

                              x =
filtered_list_broad_def_func[0].split('\t')
                              y =
filtered_list_broad_def_func[1].split('\t')

                              x[2] = float(x[2])
                              y[2] = float(y[2])

                              if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):

                                   #print 'PNT'
                                   fo1.write('PNT\n')

                              else:

                                   #print 'PN'
                                   fo1.write('PN\n')

          elif len(filtered_list_mol_func) > 5:
               #print '3 or more'
               a = filtered_list_mol_func[3].split('\t')
               b = filtered_list_mol_func[4].split('\t')
               c = filtered_list_mol_func[5].split('\t')

               a[2] = float(a[2])
               b[2] = float(b[2])
               c[2] = float(c[2])


               if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90) or ((c[0].startswith('G') or c[0].startswith('7')) and c[2] >
90):

                    #print 'PG'
#                   fo1.write('PG\n')

                    if (a[0].startswith('7')):
                         fo1.write(str(a[0]))
                         fo1.write('\n')


                    elif (b[0].startswith('7')):
                         fo1.write(str(b[0]))
                         fo1.write('\n')

                    elif (c[0].startswith('7')):
```

```
                              fo1.write(str(c[0]))
                              fo1.write('\n')


                         else:
                              fo1.write('PG\n')



                    else:

                         if len(filtered_list_broad_def_func) == 1:
                              x =
filtered_list_broad_def_func[0].split('\t')
                              x[2] = float(x[2])

                              if x[0].startswith('Transmembrane') and
x[2] > 90:

                                   #print 'PNT'
                                   fo1.write('PNT\n')

                              else:

                                   #print 'PN'
                                   fo1.write('PN\n')

                         elif len(filtered_list_broad_def_func) > 1:

                              x =
filtered_list_broad_def_func[0].split('\t')
                              y =
filtered_list_broad_def_func[1].split('\t')

                              x[2] = float(x[2])
                              y[2] = float(y[2])

                              if (x[0].startswith('Transmembrane') and
x[2] > 90) or (y[0].startswith('Transmembrane') and y[2] > 90):

                                   #print 'PNT'
                                   fo1.write('PNT\n')

                              else:

                                   #print 'PN'
                                   fo1.write('PN\n')



     #Only Molecular Function is present
```

```
     elif ('Molecular Function:' in seq_list) and ('Biological
process:' not in seq_list) and ('Broadly Defined Function:' not in
seq_list):


          if len(seq_list) == 4:
               #print 'only 1'
               a = seq_list[3].split('\t')
               a[2] = float(a[2])

               if (a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90:


                    #fo1.write('PG\n')
                    if (a[0].startswith('7')):
                         fo1.write(str(a[0]))
                         fo1.write('\n')

                    else:
                         fo1.write('PG\n')

               else:

                    fo1.write('PN\n')



          elif len(seq_list) == 5:
               #print 'only 2'
               a = seq_list[3].split('\t')
               b = seq_list[4].split('\t')

               a[2] = float(a[2])
               b[2] = float(b[2])

               if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90):

                    #print 'PG'
                    #fo1.write('PG')


                    if (a[0].startswith('7')):
                                   fo1.write(str(a[0]))
                                   fo1.write('\n')
                    elif (b[0].startswith('7')):
                                   fo1.write(str(b[0]))
                                   fo1.write('\n')

                    else:
```

```
                        fo1.write('PG\n')


                else:

                        #print 'PN'
                        fo1.write('PN\n')


        elif len(seq_list) > 5:
                #print '3 or more'
                a = seq_list[3].split('\t')
                b = seq_list[4].split('\t')
                c = seq_list[5].split('\t')

                a[2] = float(a[2])
                b[2] = float(b[2])
                c[2] = float(c[2])


                if ((a[0].startswith('G') or a[0].startswith('7')) and
a[2] > 90) or ((b[0].startswith('G') or b[0].startswith('7')) and b[2]
> 90) or ((c[0].startswith('G') or c[0].startswith('7')) and c[2] >
90):
                        #print 'PG'
                        #fo1.write('PG\n')
                        if (a[0].startswith('7')):
                                        fo1.write(str(a[0]))
                                        fo1.write('\n')
                        elif (b[0].startswith('7')):
                                        fo1.write(str(b[0]))
                                        fo1.write('\n')
                        elif (c[0].startswith('7')):
                                        fo1.write(str(c[0]))
                                        fo1.write('\n')

                        else:
                                fo1.write('PG\n')

                else:

                        #print 'PN'
                        fo1.write('PN\n')


    elif 'Molecular Function:' not in seq_list :

        #print 'no molecular'
        fo1.write('no molecular\n')


    else:
```

```
            fo1.write('problem\n')


fo1.close()
```

GPCR-CA

```
import re
from mechanize import Browser

infile = open('500_gpcr3.fasta', 'r')
lines = infile.readlines()
seqs = []
tempseq = ''
for line in lines:
    if line[0] == '>' and not tempseq:
        tempheader = line.strip()
    elif line[0] == '>':
        seqs.append([tempheader,tempseq])
        tempheader = line.strip()
        tempseq = ''
    else:
        tempseq = tempseq + line.strip()

seqs.append([tempheader,tempseq])
print seqs

fo = open("111result_500_gpcr3.csv","a")
for s in seqs:
    browser = Browser()
    browser.open("http://www.jci-bioinfo.cn/GPCR-GIA")

    browser.select_form('form1')

    browser.select_form(nr=0)
    browser['protein'] = s[1]

    response = browser.submit()

    content = response.read().decode('utf-8')

    r = content.split('<td width="62%" align="left" valign="bottom"
class="predresult">')
    r1 = r[1].split('</td>')
    fo.write(s[0])
    fo.write(',')
    fo.write(r1[0])
    fo.write('\n')
fo.close()
```

APPENDIX M: RECEPTORS IN GPCRDB AND GPCR-PENDB

Class A	
Receptors that are not in GPCRdb but available in GPCR-PEnDB	
• Ultraviolet-sensitive opsin • Putative tyramine receptor • Putative vomeronasal receptor-like protein • Putative gustatory receptor clone • Putative G-protein coupled receptor • Putative gonadotropin-releasing hormone II receptor • Putative violet-sensitive opsin • Retinochrome • Tyramine/octopamine receptor • viral G-protein coupled receptor • Violet-sensitive opsin • Vomeronasal type-1 receptor • Tyramine receptor • Red-sensitive opsin- • Putative olfactory receptor • Putative neuropeptide Y receptor • Tachykinin-like peptides receptor	• Ocellar opsin • Octopamine receptor, 1,2, beta-1R,-2R,-3R, Oamb • Octopressin receptor • Odorant receptor 131 • Olfactory receptor • Protein trapped in endoderm-1 • Protein UL78 • Pyrokinin-1 receptor • QRFP-like peptide receptor • RPE-retinal G protein-coupled receptor • RYamide receptor • Sex peptide receptor • Trissin receptor • Vertebrate ancient opsin • Visual pigment-like receptor peropsin • Serpentine receptor

Class B1 (Secretin)		
Both in GPCRdb and GPCR-PEnDB	Not in GPCRdb	Not in GPCR-PEnDB
• Calcitonin gene-related peptide type 1 receptor • Calcitonin receptor • Corticotropin-releasing factor receptor 1,2 • Gastric inhibitory polypeptide receptor • Glucagon receptor • Glucagon-like peptide 1 receptor • Growth hormone-releasing hormone receptor • Parathyroid hormone 2 receptor • Parathyroid hormone/parathyroid hormone-related peptide receptor • Pituitary adenylate cyclase-activating polypeptide type I receptor • Secretin receptor • Vasoactive intestinal polypeptide receptor	• Calcitonin receptor-like protein 1 • Diuretic hormone receptor (DH-R) • G-protein coupled receptor 157 • G-protein coupled receptor Mth, Mth2 (Protein methuselah) • G-protein coupled receptor seb-2 • Latrophilin Cirl • Latrophilin receptor-like protein A	• ADCYAP receptor type I • Adenylate cyclase activating polypeptide 1 (Pituitary) receptor type I • calcitonin gene-related peptide type 1 receptor • Calcitonin receptor, isoforms, types, fragments, like-receptors • Corticotropin releasing hormone receptor 1,2 • Deleted • G protein-coupled pituitary GHRH receptor (Growth hormone-releasing hormone receptor) • Gastric inhibitory polypeptide receptor • Glucagon like peptide 1,2 receptor • Glucagon receptor • Parathyroid hormone 1,2 receptor • parathyroid hormone/parathyroid hormone-related peptide receptor, isoforms • Secretin receptor

Vasoactive intestinal polypeptide receptor 2	• Latrophilin-like protein 1,2 • PDF receptor (Pigment-dispersing factor receptor) (Protein groom-of-PDF) • Probable G-protein coupled receptor Mth-like 1-14	• Uncharacterized protein (290) • Vasoactive intestinal peptide receptor 1,2 • Vasoactive intestinal polypeptide receptor 1,2 • VPAC1 receptor

Class B2 (Adhesion)			
Both in GPCRdb and GPCR-PEnDB	Not in GPCRdb	Not in GPCR-PEnDB	
• Adhesion G protein-coupled receptor A1-A3 (G-protein coupled receptor 123-125) • B1-B3 • D1-D2 • E1-E4 • F1-F5 • G1-G7 • L1-L4 • V1 • G-type receptor 1-3 • (CD antigen CD97) • Putative adhesion G protein-coupled receptor E4P • Cadherin EGF LAG seven-pass G-type receptor 1-3	• • Adhesion G protein-coupled receptor A1 (G-protein coupled receptor 123) *Mouse* • B1 (Brain-specific angiogenesis inhibitor 1) *Rat, Mouse* • D1 (G-protein coupled receptor 133) *Mouse* • E2 (EGF-like module receptor 2) *Dog* • F2 (G-protein coupled receptor 111) *Rat, mouse*	• A1-A3	• Other species except human, zebra fish, mouse, rat
		• B1-B3	• ||
		• D1-D2	• ||
		• E1-E4	• ||
		• F1-F5	• ||
		• G1-G7	• ||
		• L1-L4	• ||
		• V1	• ||
		• Deleted	•
		• latrophilin-1,2,3, isoforms	•
		• probable G-protein coupled receptor 113, 125	•
		• Protocadherin-like wing polarity protein stan	•
		• Unreviewed proteins	•

Class C (Metabotropic Glutamate)		
Both in GPCRdb and GPCR-PEnDB	Not in GPCRdb	Not in GPCR-PEnDB
• Extracellular calcium-sensing receptor (CaSR) • Gamma-aminobutyric acid type B receptor subunit 1,2 • G-protein coupled receptor family C group 6 member A, B, C, D • Metabotropic glutamate receptor 1-8 • Probable G-protein coupled receptor 156, 158, 179 • Probable metabotropic glutamate receptor mgl-1 • Retinoic acid-induced protein 3 • Taste receptor type 1 member 1,2,3 (G-protein coupled receptor 70)	• Gamma-aminobutyric acid type B receptor subunit 1,2 (not for C elegans) • G-protein coupled receptor family C group 6 member A (don't have from bovine goldfish and zebrafish) • (Odorant receptor 5.24), (Odorant receptor ZO6) o *Bovine* • Probable G-protein coupled receptor CG31760 • Protein bride of sevenless • Vomeronasal type-2 receptor 1, 26, 116 • Metabotropic glutamate receptor (DmGluRA) • Metabotropic glutamate receptor-like protein A-R	• Calcium polyvalent cation receptor 1 (extracellular calcium-sensing receptor isoform X1) • Calcium sensing receptor • CASR isoform 1 (CASR isoform 2) (Calcium sensing receptor) • Deleted • Extracellular calcium sensing receptor • G protein-coupled receptor class C group 6 member A, isoform X1-X3 • Chimpanzee, • Horse, • Green anole, American chameleon, • Little brown bat, • Western lowland gorilla, • Dog, • Wild turkey, • Mallard, Anas boschas, Zebra fish • Gabbr2 protein, isoform 1 • Gamma-aminobutyric acid (GABA) B receptor, 2 • Glutamate metabotropic receptor 1-8, Glutamate receptor, metabotropic 1-8b, isoforms, • G-protein coupled receptor T1R3 • GRM proteins and isoforms (1-7), • LOC100127664 protein, • LOW QUALITY PROTEIN: metabotropic glutamate receptor 1, 6 • metabotropic glutamate receptor 1-8, isoforms, fragments • Taste 1 receptor member 1-3, types, isoforms • Uncharacterized protein (261)

Class F (Frizzled)	
Both in GPCRdb and GPCR-PEnDB	Not in GPCRdb
• Frizzled-1 (Fz-1) (cFz-1) • Frizzled-1 (Fz-1) (hFz1) (FzE1) • Frizzled-1 (Fz-1) (mFz1) • Frizzled-1 (Fz-1) (rFz1) • Frizzled-1 (Fz-1) (Xfz1) • Frizzled-10 (Fz-10) (CD antigen CD350) • Frizzled-10 (Fz-10) (cFz-10) • Frizzled-10 (Fz-10) (hFz10) (FzE7) (CD antigen CD350) • Frizzled-2 (Fz-2) (cFz-2) (Fragment)	• Frizzled (Frizzled-1) (dFz1) • Frizzled (Frizzled-1) (dFz1) • Frizzled and smoothened-like protein A • Frizzled and smoothened-like protein B • Frizzled and smoothened-like protein C • Frizzled and smoothened-like protein D • Frizzled and smoothened-like protein E • Frizzled and smoothened-like protein F • Frizzled and smoothened-like protein G • Frizzled and smoothened-like protein H

• Frizzled-2 (Fz-2) (hFz2) (FzE2) • Frizzled-2 (Fz-2) (mFz2) (Frizzled-10) (Fz-10) (mFz10) • Frizzled-2 (Fz-2) (rFz2) • Frizzled-2 (Fz-2) (Xfz2) • Frizzled-3 (Fz-3) (hFz3) • Frizzled-3 (Fz-3) (mFz3) • Frizzled-3 (Fz-3) (Xfz3) • Frizzled-4 (Fz-4) (cFz-4) • Frizzled-4 (Fz-4) (hFz4) (FzE4) (CD antigen CD344) • Frizzled-4 (Fz-4) (mFz4) (CD antigen CD344) • Frizzled-4 (Fz-4) (rFz4) (CD antigen CD344) • Frizzled-4 (Fz-4) (Xfz4) • Frizzled-5 (Fz-5) • Frizzled-5 (Fz-5) (hFz5) (FzE5) • Frizzled-5 (Fz-5) (mFz5) • Frizzled-5 (Fz-5) (Xfz5) • Frizzled-6 (Fz-6) • Frizzled-6 (Fz-6) • Frizzled-6 (Fz-6) (hFz6) • Frizzled-6 (Fz-6) (mFz6) • Frizzled-7 (Frz7) (Fz-7) • Frizzled-7 (Fz-7) (cFz-7) • Frizzled-7 (Fz-7) (hFz7) (FzE3) • Frizzled-7 (Fz-7) (mFz7) • Frizzled-8 (Fz-8) • Frizzled-8 (Fz-8) (hFz8) • Frizzled-8 (Fz-8) (mFz8) • Frizzled-8 (Fz-8) (Xfz8) • Frizzled-9 (Frizzled-like protein 9) (rFz9) • Frizzled-9 (Fz-9) (cFz-9) (Fragment) • Frizzled-9 (Fz-9) (hFz9) (FzE6) (CD antigen CD349) • Frizzled-9 (Fz-9) (mFz3) (mFz9) (CD antigen CD349) • Protein smoothened (SMOH) (Smooth) (dSMO) • Smoothened homolog (SMO) • Smoothened homolog (SMO) • Smoothened homolog (SMO) (Fragment) • Smoothened homolog (SMO) (Protein Gx)	• Frizzled and smoothened-like protein J (Cell number regulator protein A) • Protein mom-5 • Frizzled and smoothened-like protein K • Frizzled and smoothened-like protein L • Frizzled and smoothened-like protein M • Frizzled and smoothened-like protein N • Frizzled and smoothened-like protein O • Frizzled and smoothened-like protein P • Frizzled and smoothened-like protein Q • Frizzled/smoothened-like sans CRD protein A • Frizzled/smoothened-like sans CRD protein B • Frizzled/smoothened-like sans CRD protein C • Frizzled/smoothened-like sans CRD protein D • Frizzled/smoothened-like sans CRD protein E • Frizzled/smoothened-like sans CRD protein F • Frizzled/smoothened-like sans CRD protein G • Frizzled/smoothened-like sans CRD protein H • Frizzled/smoothened-like sans CRD protein J • Frizzled-10-A (Fz-10A) (Xfz10-A) • Frizzled-10-B (Fz-10B) (Xfz10-B) (Frizzled-9) (Fz-9) (Xfz9) • Frizzled-2 • Frizzled-2 (dFz2) • Frizzled-3 (dFz3) • Frizzled-3 (Fz-3) (cFz-3) (Fragment) • Frizzled-4 (dFz4) • Frizzled-6 (Fz-6) (cFz-6) (Fragment) • Frizzled-7-A (Fz-7-A) (Xfz7-A) • Frizzled-7-B (Fz-7-B) (Xfz7-B) • Frizzled-8 (Fz-8) (cFz-8) (Fragment) • Protein mom-5

Taste 2 Receptors	
Both in GPCRdb and GPCR-PEnDB	Not in GPCRdb
• Taste receptor type 2 member 1, 3, 4, 5, 7, 8, 9, 10, 13, 14, 16, 19, 20, 30, 31, 38, 39, 40, 41, 42, 43, 45, 46, 50, 60, 70, 103, 105, 106, 107, 109, 110, 113, 114, 119, 123, 140, 143,	• Taste receptor type 2 member 40, 62, 64, 66, 102, 103, 104, 105, 106, 107, 109, 110, 113, 114, 116, 117, 120, 123, 124, 125, 129, 134, 135, 136, 143

Curriculum Vita

Khodeza Begum was born in Dhaka, Bangladesh. The youngest daughter of Md. Khorshed Alam and Monowara Begum, she got her secondary and higher secondary school certificates in 2005 and 2007 respectively from Viqarunnisa noon school and college in Dhaka, Bangladesh. She entered The American International University of Bangladesh in fall 2008 to pursue her Bachelor of Science degree in Electrical and Electronics Engineering. She received 'Summa Cum Laude' for academic excellence and graduated in 2012. After pursuing her bachelor's degree, she started working as a lab instructor in the engineering department of North South University in Dhaka, Bangladesh for over a year where she taught programming languages and computer networking lab. In 2014, she worked as a lab officer in Management Information System of the Business school in North South University and later she joined the Information technology department as an IT officer. Khodeza got accepted in the Computational Science program at University of Texas at El Paso to pursue her Doctoral degree in fall 2015 and earned her Master of Science degree in 2017, she currently resides in El Paso with her husband and one child.

Khodeza's Dissertation, "GPCR-PEnDB: A database of protein sequences and derived features to facilitate prediction and classification of G protein-coupled receptor" was supervised by Dr. Ming-Ying Leung.

Email address: kbegum@miners.utep.edu