University of Texas at El Paso

ScholarWorks@UTEP

Open Access Theses & Dissertations

2020-01-01

Prediction And Classification Of G Protein-Coupled Receptors Using Statistical And Machine Learning Algorithms

Fredrick Ayivor
University of Texas at El Paso

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

Part of the Bioinformatics Commons

Recommended Citation

Ayivor, Fredrick, "Prediction And Classification Of G Protein-Coupled Receptors Using Statistical And Machine Learning Algorithms" (2020). *Open Access Theses & Dissertations*. 2927.
https://scholarworks.utep.edu/open_etd/2927

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.

PREDICTION AND CLASSIFICATION OF G PROTEIN-COUPLED RECEPTORS USING STATISTICAL AND MACHINE LEARNING ALGORITHMS

FREDRICK AYIVOR

Doctoral Program in Computational Science

APPROVED:

Ming-Ying Leung, Ph.D., Chair

Charlotte M. Vines, Ph.D.

Thompson Sarkodie-Gyan, Ph.D.

Jonathon E. Mohl, Ph.D.

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

©Copyright

by

FREDRICK AYIVOR

2020

PREDICTION AND CLASSIFICATION OF G PROTEIN-COUPLED RECEPTORS USING STATISTICAL AND MACHINE LEARNING ALGORITHMS

by

FREDRICK AYIVOR, 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

May 2020

Acknowledgements

I would like to express my deep-felt gratitude to my advisor, Dr Ming-Ying Leung of the Computational Science Department at The University of Texas at El Paso, for her advice, encouragement, enduring patience and constant support. She was never ceasing in her belief in me, always providing clear explanations when I was lost, constantly driving me with energy when I was tired, and always, *always* giving me her time, in spite of anything else that was going on.

I also wish to thank the other members of my committee, Dr. Charlotte Vines of the Biological Sciences Department, Dr. Jonathon E. Mohl of the BBRC and Dr. Sarkodie-Gyan Thompson of the Electrical & Computer Engineering all at The University of Texas at El Paso. Their suggestions, comments and additional guidance were invaluable to the completion of this work.

My heartfelt gratitude also goes to the research group members; Jonathan Mohl, Khodeza Begum and Eder Perez for their enormous participation in putting together this work.

Also not forgetting the tremendous support of Grant 5G12MD007592 to the Border Biomedical Research Center from the National Institute on Minority Health and Health Disparities, a component of the National Institutes of Health.

Additionally, I want to thank The University of Texas at El Paso Computational Science Department professors and staff for all their hard work and dedication, providing me the means to complete my degree.

And finally, I must thank my dear wife for putting up with me during the development of this work with continuing, loving support and no complaint. I do not have the words to express all my feelings here, only that I love you, Gladys V. Matiedje Ayivor!

Abstract

G protein-coupled receptors (GPCRs) are transmembrane proteins with important functions in signal transduction and often serve as therapeutic drug targets. With increasing availability of protein sequence information, there is much interest in computationally predicting GPCRs and classifying them to indicate their possible biological roles. Such predictions are cost-efficient and can be valuable guides for designing wet lab experiments to help elucidate signaling pathways and expedite drug discovery. There are existing computational tools for GPCR prediction and classification that involve statistical and machine learning approaches such as principal component analysis, support vector machines, hidden Markov models, etc. These tools use protein sequence derived features including amino acid and dipeptide compositions and other autocorrelation descriptors of physicochemical properties. While prediction accuracies of over 90% were generally reported for their own test data, no direct performance comparison of the different tools has been conducted using a unified test dataset. Furthermore, their abilities in distinguishing GPCRs from transmembrane non-GPCRs have not been measured, and none of the existing tools has the capability of fully classifying a general GPCR down to the subtype level.

In this dissertation, I proposed two new methods, the penalized multinomial logistic regression (Log-Reg) algorithm and the multi-layer perceptron neural network (MLP-NN) to address this multilayer problem of GPCR prediction and classification using 1360 sequence features. Training and testing were conducted uniformly with a test dataset containing 2016 confirmed GPCRs, and 3100 negative examples including transmembrane non-GPCRs. To assess our new methods, their performance were compared with two available tools, GPCR-pred and PCA-GPCR. Both Log-Reg and MLP-NN substantially reduced the false positive rates in distinguishing GPCRs from transmembrane non-GPCRs. They also produced highly accurate GPCR classification results down to the subtype level with average accuracies in the 96-99% range. Furthermore, we applied feature reduction techniques to generate a non-redundant feature set to increase the computational efficiency for Log-Reg and MLP-NN with little impact on accuracy. These algorithms have been implemented as Python programs and are being incorporated into the web server gpcr.utep.edu, which can be accessed by GPCR researchers worldwide.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In biology and biochemistry the term "receptor" refers to a class of cellular macromolecules that are specifically or directly involved in chemical signaling between and within cells [Cooper et al., 2000]. As a receptor, it does not only recognize the particular molecules that activate it [Venkatakrishnan et al., 2019] but also responds by stimulating the cellular pathways. Therefore, the interaction of a hormone, neurotransmitter, or intracellular messenger with its receptor(s) may lead to a change in cellular activity.

Among the cell surface receptors, the G protein-coupled receptors (GPCRs) occupy a special place of importance as they are a large protein family sharing a unifying signal transduction mechanism. The number of identified GPCRs has been increasing rapidly. It is estimated that more than a thousand different GPCRs exist in mammals, thus constituting one of the largest protein families in nature [Hauser et al., 2017]. A wide range of neurotransmitters, neuropeptides, polypeptide hormones, inflammatory mediators, and other bioactive molecules transmit their signals to the intracellular environment by specific interactions with GPCRs that would in turn couple with GTP-binding proteins (G protein) for activation of intracellular effector systems [Tuteja, 2009].

1.1 GPCR characteristics and classification

GPCRs have a seven-helix-bundled structure embedded in the cell membrane. A GPCR molecule consists of a single polypeptide chain of variable length that traverses the lipid bilayer seven times as shown in figure 1.1, forming characteristic transmembrane helices and alternating extracellular and intracellular sequences [Ulloa-Aguirre et al., 2014].

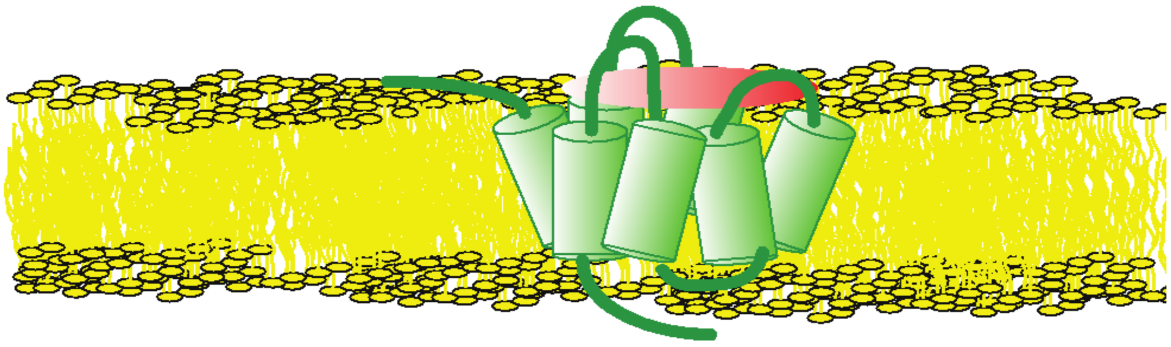


Figure 1.1: G-protein-coupled receptor with ligand [C. Vines, Personal Communication, December 2017]

GPCRs regulate many important physiological processes including automatic nervous system transmission, sense of smell, and regulation of immune system activity. They are the largest known class of cell surface receptors [Strader et al., 1994] and represent 1% of the total mammalian genes [Barreda-Gomez et al., 2010]. GPCRs can thus play roles in secretion, proliferation, chemotaxis, heart rate, and neuro-transmission [Spiegel et al., 1992]. GPCR upon activation by a ligand/agonist undergoes a conformational change and then activate the heterotrimeric G proteins (G_α, G_β and G_γ) which makes the G_α subunit to exchange guanine diphosphate (GDP) for guanine triphosphate (GTP) [Tuteja, 2009]. This results in the dissociation of G_β/G_γ dimer from the G_α, initiating the intracellular signaling responses [Tuteja, 2009]. The binding interactions of these receptors with G proteins have been investigated by means of structural bioinformatics [Chou 2005a].

Since GPCRs are a very large family, classifying them hierarchically into smaller groups would be very helpful in understanding their individual biological roles and functional pathways. One of the most frequently used systems is IUPHAR (International Union of Basic and Clinical Pharmacology) classification [Horn et al., 2003] that divide GPCRs into six major categories named Class (Family) A, B, C, D, E, and F based on sequence homology and functional similarity. Then subclasses (or subfamilies) are assigned using Roman number nomenclature [Attwood and Findlay 1994; Kolakowski, 1994]. The IUPHAR classification is designed to cover all GPCRs in both vertebrates and invertebrates. However, some families, namely Class D and Class E do not exist in humans. This is why the GRAFS classification system, designed towards mammalian species, has been devised. The GRAFS system clusters GPCRs in five main families that we term glutamate (G), rhodopsin (R), adhesion (A), frizzled/taste2 (F), and secretin (S). [Fredriksson and Lagerstrom 2003].

In this work, I have combined IUPHAR and GRAFS systems to form a 5-level classification system for any given protein. At Level 1 (highest level) of this combined system, a protein is classified to be in the GPCR superfamily or not. If it is a GPCR, it is further classified into eight separate families at level 2. Each family is further divided into subfamilies at the third level, followed by the fourth and fifth levels of sub-subfamilies and subtypes.

1.2 GPCR prediction and classification tools

With the advances in next generation sequencing technologies, protein sequence data is becoming abundantly available, but the understanding of each protein's biological roles is lagging behind. Developing computational approaches for predicting whether a protein sequence is a GPCR and classifying GPCRs based only on their amino acid sequence information have become a research area of great interest because GPCR classification plays a crucial role in identifying the specific functions of GPCRs. As each GPCR subtype has its own characteristic ligand-binding property, coupling partners of trimeric G-proteins, and interaction partners of oligomerization [Kristiansen 2004], prediction of GPCRs at the subtype level becomes significant in the effort to decipher the functions of individual GPCRs. GPCR classification, however, is a challenging task. It involves five levels and except for Level 1 (Superfamily), each level presents a multinomial classification problem. Fortunately, as more confirmed GPCR sequences are being accumulated, it becomes possible to tackle this multi-level multinomial classification problem as my dissertation research.

A number of web-based tools have been developed by different groups from various computational methods for GPCR prediction and classification, including GPCRTm [Rios et al., 2015], GPCRpred [Bhasin et al., 2004], PCA-GPCR [Peng et al., 2010], PredGPCR [Papasaikas et al., 2004], SVMProt [Cai et al., 2003], GPCRGIA [Lin et al., 2009] and GPCRCA [Xiao et al., 2009].

However, there are several major limitations and shortcomings that need to be overcome:

1. These programs generally show high false positive rates in identifying GPCRs when tested with a collection of transmembrane proteins containing a mixture of GPCRs and transmembrane non-GPCRs.

2. Many algorithms are published but the actual programs are either not currently available, or cannot be used for large scale testing [Karchin et al., 2002; Cai et al., 2003; Peng et al. 2010].

3. No direct comparison of different algorithms with the same test dataset has been conducted. The tools listed above each used their own datasets for accuracy assessment.

1.3 Research objectives

The overall goal of my research is to analyze the discriminative power of the existing methodologies for GPCR prediction and classification, and to devise accurate and efficient algorithms that can be widely used by scientists for GPCR research. After reviewing current literature of computational algorithms for GPCR prediction and classification in Chapter 2, I will describe the work done to accomplish the following specific aims in the subsequent chapters of this dissertation.

1. Identify optimal computational approaches for GPCR prediction

   By surveying the performance, in terms of both accuracy and efficiency, of current state-of-the-art statistical and machine learning algorithms, I identified optimal computational approaches suitable for analyzing GPCRs with the aim of utilizing them to address the GPCR classification problem below.

2. Address the multi-level GPCR classification problem

   Existing GPCR prediction and classification tools are generally not able to accurately classify protein sequences all the way from Level 1 (superfamily) to Level 5 (subtype). This is likely caused by the biased data used in training the classification algorithms, as the vast majority of GPCR sequences are in Class A while examples in Classes D and E are scanty. Using the optimal computational approaches identified in Objective 1 above and the data collection in the GPCR-PEnDB developed by our group [Begum et al., 2020, http:// gpcr.utep.edu/], I developed new methods to handle such data imbalance issues in order to improve the GPCR classification accuracies and enhance the computational efficiency by reducing the feature set through the use of ranked based methods.

   Finally, I investigated the possibilities of further improving the classification accuracies and execution time by reducing the feature set through the use of ranked based methods.

3. Software implementation and distribution.

   The final classification methods were then developed into software packages and will be incorporated into our GPCR-PEn webserver (https:// gpcr.utep.edu) and distributed via the UTEP bioinformatics data repository so that it can be accessed by GPCR researchers worldwide. The source code of my program will also be deposited in GitHub (https://github.com/fayivor/Prediction_tools) so that it can be downloaded and modified by other computational scientists for further improvements or adapted for other purposes.

Chapter 2

Literature Review

This chapter accounts for the various approaches and algorithms related to the classification of GPCRs into different levels. The initial sections highlight an overview of GPCRs and their derived features. This is followed by a discussion of computational algorithms which use machine learning and statistical approaches for the classification and prediction. It also entails the accuracies by the various algorithms. The final part covers the combination of different methods.

2.1 Overview of GPCRs and their classification

GPCRs as membrane proteins, are very difficult to crystallize and most of them will not dissolve in normal solvents [Xiao et al., 2009]. Contrary to that, the amino acid sequences of about 3200 confirmed GPCRs are known with the rapid accumulation of new protein sequence data produced by high-throughput sequencing technology.

GPCR proteins can be grouped in hierarchical structure with five levels (Figure 2.1). The figure shows the combined IUPHAR and GRAFS systems used in this work to form a 5-level [Begum et al., 2020] classification system.

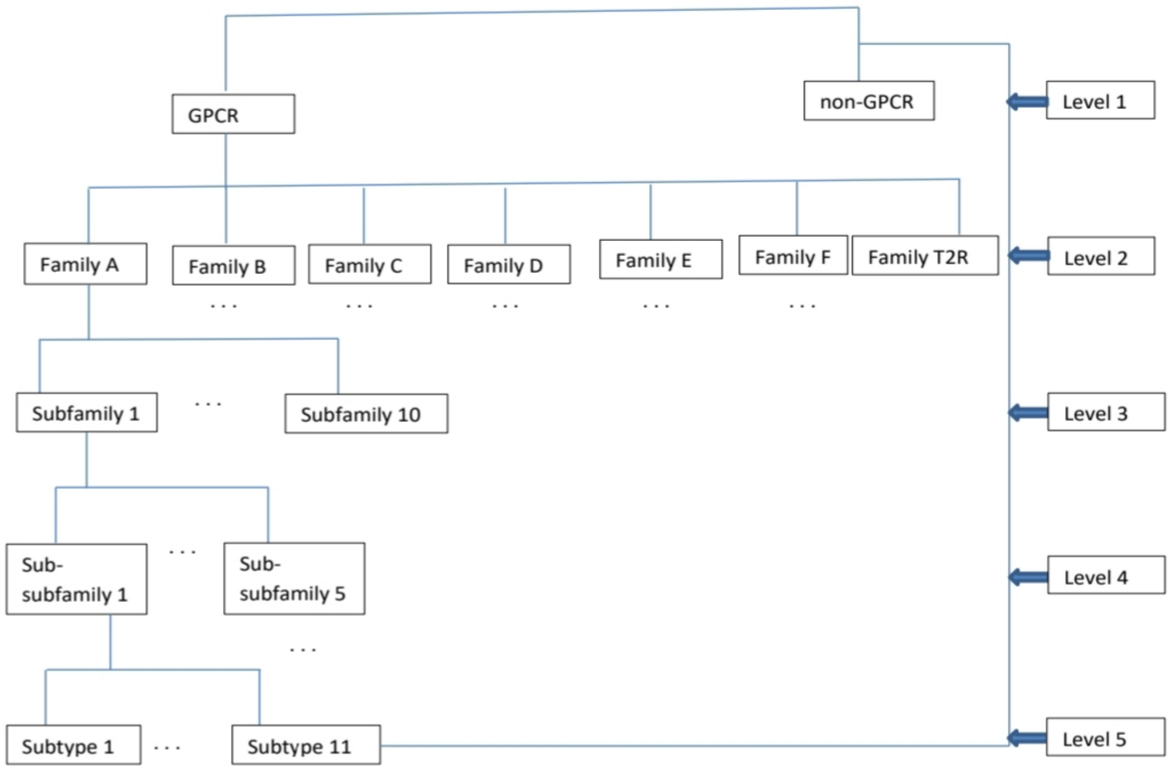


Figure 2.1: The hierarchical classification structure for GPCRs.

The family A, known as rhodopsin-like receptors, comprise 80% of all the GPCRs and can transduce a range of stimuli including peptide hormone, light, nucleotides, and chemokines [Bryson-Richardson et al., 2004]. The human non-olfactory receptors of rhodopsin-like class bind peptides and biogenic amines [Fridmanis et al., 2006]. There are 11 unique subfamilies, 61 unique sub-subfamilies and 287 unique subtypes under A. This means there are 287 uique groups for Class A GPCR [GPCRdb].

Family B1 GPCR (secretin-like receptors) is composed of 1 subfamily (peptide), 5 sub-subfamilies and 15 subtypes. This implies that, there are 15 unique members [GPCRdb] and it includes important receptors such as vasoactive intestinal peptide receptors (VPAC), pituitary adenylyl cyclase activating peptide receptor (PAC1R), secretin receptor (SECR), growth hormone releasing factor receptor (GRFR), glucagon receptor (GCGR), glucagon like-peptide 1 and 2 receptors (GLPR), gastric inhibitory peptide receptor (GIPR) [Laburthe et al., 2007]. They represent very important targets for the development of drugs having therapeutical impact on many diseases such as diabetes, chronic inflammation, stress, neurodegeneration, and osteoporosis. These GPCRs also interact with a few accessory proteins which play a role in cell signaling, receptor expression and/or pharmacological profiles of receptors [Couvineau et al., 2012a].

Family B2 consists of a large number of family-B GPCRs with long extracellular amino termini, containing diverse structural elements, linked to the core 7TM motif. Members of this family have been given various names: EGF-TM7 receptors, to reflect the presence of epidermal growth factor (EGF) domains within the amino-terminal regions of many of these proteins [McKnight et al., 1996]; LN-TM7 receptors, denoting seven-transmembrane proteins with a long amino terminus [Zendman et al., 1999]; and LNB-TM7 receptors, to denote 'long amino terminus, family B' [Stacey et al., 2000]. There is 1 unique subfamily known as Adhesion, 9 unique sub-subfamilies and 33 subtypes. In short, there are 33 unique members under this family [GPCRdb].

The characteristic feature of all family-B GPCRs (B1/ B2) is the 7TM motif, which is distantly related to comparable regions of some other GPCR families but much more highly conserved within family B. Conserved cysteine residues within extracellular loops EC1 and EC2 probably form a disulphide bridge, by analogy with family-A GPCRs, in which this feature is also conserved [Palczewski et al., 2000]. In contrast to family-A GPCRs, however, many of which appear to rely on internal hydrophobic sequences for targeting to the plasma membrane, most family-B GPCRs appear to have an amino-terminal signal peptide [Harmar et al., 2001].

Family C GPCRs have nutrients like amino acids, ions and sugar molecules as their endogenous agonists. This family consists of eight metabotropic glutamate (mGlu) receptors, a calcium-sensing receptor (CaR), two γ-aminobutyric acid$_B$ (GABA$_B$) receptors, the promiscuous L-α-amino acid receptor GPRC6A, families of taste and pheromone receptors, and five orphan receptors [Bettler et al., 2004]. The family C GPCRs are also involved in important physiological processes throughout the body, the mGlu receptors being localised almost exclusively in the CNS, whereas the other family C receptors are expressed both centrally and in peripheral tissues [Bräuner-Osborne 2007]. The eight mGlu receptor subtypes cloned to date are divided into three subgroups based on amino acid sequence similarity, agonist pharmacology and G-protein coupling property: group I being comprised of mGlu$_1$ and mGlu$_5$, group II of mGlu$_2$ and mGlu$_3$, and group III containing the remaining four subtypes [Jensen 2004].

Fungal pheromone mating factor receptors form a distinct family of GPCRs and are also known as Class D GPCRs. The STE2 and STE3 receptors of Class D are integral membrane proteins that may be involved in the response to mating factors on the cell membrane [Nakayama et al., 1985]. For example, in the widely used model yeast Saccharomyces cerevisiae, mating is controlled by GPCR-mediated perception of both peptide pheromones and nutritional status. Pheromone receptors are a small class of GPCRs used for chemical communication by the organisms and play a role in controlling interactions between individuals of a single species [Martini et al., 2001]. Fungal GPCRs are extremely diverse in the environmental signals that they detect, including hormones, proteins, nutrients, ions, hydrophobic surfaces and light [Kochman, 2014]. Fungal GPCRs do not belong to any of the mammalian receptor classes, making them fungal-specific targets to intervene in fungal disease and mycotoxin contamination [Dijck, 2009].

Family E, cyclic AMP (known as cAMP) receptors contribute towards chemotactic signaling system of slime molds [Prabhu et al., 2006]. In Dictyostelium discoideum, the cyclic AMP receptors coordinate aggregation of individual cells into a multicellular organism, and regulate the expression of a large number of developmentally-regulated genes.

Family F, are called the Frizzled/Smoothened proteins. Smoothened in humans is encoded by the SMO gene. Smoothened is a component of the hedgehog signaling pathway and is conserved from flies to humans. It is the molecular target of the natural teratogen cyclopamine [Ruiz-Gómez et al., 2007]. Frizzled [Malbon CC, 2004] also serves as receptors in the Wnt signaling pathway and other signaling pathways. When activated, Frizzled leads to activation of Dishevelled in the cytosol. There are 11 unique subtypes which are FZD_1-FZD_{10} and SMO.

Family Taste 2 receptors (T2R) function as chemoreceptors that interact with taste stimuli to initiate an afferent signal transmitted to the brain, which results in taste perception. Because many taste ligands do not easily permeate cell membranes, taste receptors are believed to be a part of the taste receptor cells (TRC) membranes. This family has a distant relationship with the rhodopsins 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. There are 25 unique subtypes.

2.2 Protein features commonly used in GPCR prediction and classification.

In the basic composition of life, proteins play a central role in most cellular functions such as gene regulation, metabolism and cell proliferation [van der Knaap et al., 2016]. As a classification problem, statistical and machine learning algorithms for protein sequence class prediction involve three main steps: i) protein feature representation; ii) algorithm selection for classification; iii) optimal feature selection [Li et al., 2014]. Among the three steps, protein feature representation (feature extraction) is the most critical factor for the success of protein structural class prediction.

Feature extraction means how to extract features from protein sequences so that each protein is represented as a fixed-length numerical vector [Peng et al., 2010]. The commonly-used feature extraction methods for proteins are based on amino acid composition [Chou et al., 2004] and dipeptide composition [Qao et al., 2006], with more complicated ones include Chou's pseudo amino acid composition [Lin et al., 2006], the cellular automaton image approach [Xiao et al., 2009], profile hidden Markov models [Papasaikas et al., 2003], fast fourier transform [Guo et al., 2006], wavelet-based time series analysis [Gupta et al., 2008] and Fisher score vectors [Karchin et al., 2002].

Among the above feature extraction methods used in the above mentioned studies, I collected 1360 descriptor values (features) which can be divided into four groups. Each group has been used as an independent descriptor (set of features) for predicting proteins and peptides of various profiles by using statistical or machine learning methods. The first group includes two descriptors, the amino acid composition and dipeptide composition, with 20 and 400 descriptor values respectively [Shepherd et al., 2003]. The second group contains 30 pseudo amino acid composition descriptor values. The third group contains two sequence-order descriptors [Chou et al., 2000]: sequence-order-coupling number with 60 descriptor values, and the other is quasi-sequence-order with 100 descriptor values. The fourth group contains three different sets of autocorrelation features: normalized Moreau-Broto autocorrelation [Reczko et al., 1994], Moran autocorrelation [Horne et al., 1998] and Geary autocorrelation [Soka et al., 2006], each set having 240 descriptor values.

Amino acid and dipeptide composition are simple descriptors of protein sequence features [Shepherd et al., 2003], which have been used for predicting protein fold and structural classes [Grassmann et al., 1999], functional classes [Bhasin et al., 2004] and subcellular locations [Hua et al., 2001] at accuracy levels of 72%-95%, 83-97%, and 79-91% respectively.

Amino acid composition is the fraction of each amino acid type in a sequence. A total of 20 descriptor values are computed for the 20 types of amino acids. Likewise, dipeptide composition is the fraction of the different dipeptides observed in the amino acid sequence. A total of 400 descriptor values are computed for the 20 $\times$ 20 amino acid combinations [Bhasin et al., 2004].

Autocorrelation features describe the level of correlation between two objects (protein or peptide sequences) in terms of their specific structural or physicochemical property [Broto et al., 1984] which are defined based on the distribution of amino acid properties along the sequence [Kawashima et al., 2000]. There are eight amino acid properties used for deriving these autocorrelation descriptors. The first is hydrophobicity scale derived from the bulk hydrophobic character for the 20 types of amino acids in 60 protein structures [Cid et al., 1992]. The second is the average flexibility index derived from the statistical average of the B-factors of each type of amino acids in the available protein X-ray crystallographic structures [Bhaskaran et al., 1988]. The third is the polarizability parameter computed from the group molar refractivity values [Charton et al., 1982]. The fourth is the free energy of amino acid solution in water [Hutchins harton et al., 1982]. The fifth is the residue accessible surface areas taken from average values from folded proteins [Chothia et al., 1976]. The sixth is the amino acid residue volumes measured by Fisher [Bigelow et al., 1967]. The seventh is the steric parameters derived from the van der Waals raddi of amino acid side-chain atoms [Charton et al., 1981]. The last one is the relative mutability obtained by multiplying the number of observed mutations by the frequency of occurrence of the individual amino acids [Dayhoff et al., 1978].

In Chapter 3 we will describe more details about how these protein features or descriptor values are calculated based on the physicochemical properties of an amino acid sequence and how these are used for GPCR prediction and classification.

2.3 Computational algorithms

In view of the extremely unbalanced state, it is vitally important to develop a computational method that can quickly and accurately predict the structure and function of GPCRs from sequence information. Many predictive methods have been developed, which in general, can be roughly divided into three categories. The first one is proteochemometric approach developed by Lapinsh [Lapinsh et al., 2005]. However, the methods need structural information of organic compounds. The second one is based on similarity searches using primary database search tools (e.g. BLAST, FASTA) and such database searches coupled with searches of pattern databases (PRINTS) [Lapinsh et al., 2002].

However, they do not seem to be sufficiently successful for comprehensive functional identification of GPCRs, since GPCRs make up a highly divergent family, and even when they are grouped according to similarity of function, their sequences share strikingly little homology or similarity to each other [Inoue et al., 2004]. The third one is based on statistical and machine learning methods, including support vector machines (SVM) [Karchin et al., 2002], hidden Markov models (HMMs) [Qian et al., 2003], covariant discriminant (CD) [Chou et al., 2002], nearest neighbor (NN) [Gao et al., 2006] and other techniques.

The machine learning algorithms can be categorized into supervised and unsupervised learning methods. Supervising a machine learning model means to observe and direct the model, by training it with some data from a labeled data set. In unsupervised learning, we do not supervise the model, but let the model work on its own to discover information that may not be visible to the human eye. That is, unsupervised model provides unlabeled data that the algorithm tries to make sense of by extracting features and patterns on its own.

2.3.1 Supervised machine learning algorithms

Support vector machines (SVMs)

SVMs are a set of related supervised learning methods used for classification, regression and novelty detection [Vapnik, 1995]. They belong to a family of generalized linear classifiers and are all about decision boundaries so do not provide posterior probabilities. A special property of SVMs is that they simultaneously minimize the empirical classification error and maximize the geometric margin; hence they are also known as maximum margin classifiers.

For a two-class classification problem, one can use linear models of the form

$$y(x) = w^T \phi(x) + b \tag{2.1}$$

where $\phi(x)$ denotes a fixed feature-space transformation, and the bias parameter b is explicit. Note that to avoid working explicitly in feature space a dual representation expressed in terms of kernel functions is used. The training data set comprises N input vectors $x_1, ..., x_N$, with corresponding target values $t_1, ..., t_N \in \{-1, 1\}$, and new data points x are classified according to the sign of $y(x)$.

Assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters w and b such that a function of the form (2.1) satisfies $y(x_n) > 0$ for points having $t_n = +1$ and $y(x_n) < 0$ for points having $t_n =$ -1, so that $t_n y(x_n) > 0$ for all training data points. The SVM approaches the problem of multiple solutions (all classify the training data set exactly) through the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples [Hastie et al., 2009].

In SVMs the decision boundary is chosen to be the one for which the margin is maximized. A simple insight into the origins of maximum margin has been given by Tong and Koller (2000) who consider a framework for classification based on a hybrid of generative and discriminative approaches. They first model the distribution over input vectors x for each class using a Parzen density estimator with Gaussian kernels [Cohen et al., 2008; Duda et al., 2000] having a common parameter σ^2. Together with the class priors, this defines an optimal misclassification-rate decision boundary. However, instead of using this optimal boundary, they determine the best hyperplane by minimizing the probability of error relative to the learned density model. In the limit $\sigma^2 \to 0$, the optimal hyperplane is shown to be the one having maximum margin. The intuition behind this result is that as σ^2 is reduced, the hyperplane is increasingly dominated by nearby data points relative to more distant ones. In the limit, the hyperplane becomes independent of data points that are not support vectors.

In an experiment conducted by Rachel et al. (2002) to classify protein sequences of the GPCR superfamily [Watson and Arkinstall, 1994], the authors specifically aimed to recognize small subfamilies of GPCRs that bind the same ligand. This requires extension of the two-class problem to a multi-class problem. They chose the simplest approach to multi-class SVMs by training k (>2) one-to-rest classifiers. An initial step was required in which each protein sequence was transformed into a fixed-length feature vector. Next a two-class SVM was trained for each GPCR subfamily.

Each of these SVMs learns to distinguish between subfamily members and non-members by making several passes through training set and using the kernel function to compute a weight for each sequence. In the results of a two-fold cross-validation experiments that compared multi-class SVMs with two popular classification methods: BLAST [Altschu et al., 1990] and profile HMMs, it was found that one of the methods performed better than SVMs at the superfamily level. However, SVMs made significantly fewer errors than the other methods when applied to discriminating subfamilies of GPCRs. SVM was computationally expensive [Bhasin & Raghava M. 2004] when compared to BLAST [Altschu et al., 1990] and hidden Markov models [Durbin et al., 1998] .

Covariant-discriminant algorithm

Linear and Quadratic Discriminant analysis (LDA/QDA) are common tools for classification problems [Srivastava S., et al., 2007; Anagnostopoulos et al. 2012]. For these methods we assume observations are normally distributed within group. We estimate a mean and covariance matrix for each group and classify using Bayes theorem. With Linear discriminant analysis, we estimate a single, pooled covariance matrix, while for QDA we estimate a separate covariance matrix for each group [Hastie et al., 2009]. Rarely do we believe in a homogeneous covariance structure between groups, but often there is insufficient data to separately estimate covariance matrices.

Consider the usual two class problem: the data consists of n observations, each observation with a known class label $\in \{1, 2\}$, and p covariates measured per observation. Let y denote the n-vector corresponding to class (with n_1 observations in class 1 and n_2 in class 2), and X, the n by p matrix of covariates. We would like to use this information to classify future observations.

We further assume that, given class $y(l)$, each observation, x_l, is independently normally distributed with some class specific mean $\mu_y(l) \in \mathbf{R}^p$ and covariance $\sum_{y(l)}$, and that $y(l)$ has prior probability π_1 of coming from class 1 and π_2 from class 2. From here we estimate the two mean vectors, covariance matrices, and prior probabilities and use these estimates with Bayes theorem to classify future observations. A number of different methods have been proposed to estimate these parameters. The simplest is QDA, which estimates parameters by their maximum likelihood estimates.

$$\pi_k = \frac{n_k}{n} \tag{2.2}$$

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{y(l)=k} x_l \tag{2.3}$$

and

$$\hat{L_k} = \frac{1}{n_k} \sum_{y(l)=k} (x_l - \mu_k)(x_l - \mu_k)^T \tag{2.4}$$

To classify a new observation x, one finds the class with the highest posterior probability. This is equivalent in the two class case to considering

$$D(x) = \log(\frac{\pi_1}{\pi_2}) - \frac{1}{2}(x - \mu_1)^T L_1^{-1}(x - \mu_1) + \frac{1}{2}(x - \mu_2)^T L_1^{-1}(x - \mu_2) + \log\det L_1^{-1/2} L_2^{1/2} \tag{2.5}$$

and if D(x) $>$ 0 then classifying to class 2, otherwise to class 1.

Linear Discriminant Analysis (LDA) is a similar but more commonly used method. It estimates the parameters by a restricted MLE — the covariance matrices in both classes are constrained to be equal. So, for LDA

$$\hat{L_1} = \hat{L_2} = \frac{1}{n} \sum_{y(l)=k} (x_l - \mu_{y(l)})(x_l - \mu_{y(l)})^T \tag{2.6}$$

While one rarely believes that the covariance matrices are exactly equal, often the decreased variance from pooling the estimates greatly outweights the increased bias.

In the study on correlation of G-protein-coupled receptors types with amino acid composition, a QDL type algorithm was used by Chou and Elrod (1999) to classify the rhodopsin-like amine GPCRs (according to the GPCRDB [Horn et al., 1998]) into six groups (acetylcholine, adrenoceptor, dopamine, histamine, serotonin and octopamine) based on their amino acid compositions. The overall Jackknife test (sequentially deleting one observation in the dataset, then recomputing the desired statistic to estimate its bias) was applied to a dataset of 167 GPCRs and the accuracy was 83.23%. This suggested that the types of GPCR were predictable to considerably accurate extent if a complete or quasi-training dataset could be established.

K-nearest neighbor (KNN)

The KNN algorithm is a robust and versatile classifier that is often used as a benchmark for more complex classifiers such as artificial neural networks (ANN) and SVM.

Let x denote a feature (predictor, attribute) and y denote the target (label, class) we are trying to predict. KNN falls in the supervised learning family of algorithms. Informally, this means that we are given a labelled dataset consisting of training observations (x, y) and would like to capture the relationship between x and y. More formally, our goal is to learn a function h:X$\mapsto$Y so that given an unseen observation x, $h(x)$ can confidently predict the corresponding output y.

The KNN classifier is also non parametric which means it makes no explicit assumptions about the functional form of h, avoiding the dangers of mismodeling the underlying distribution of the data. As an instance-based learning algorithm meaning, KNN does not explicitly learn a model, instead it chooses to memorize the training instances which are subsequently used as “knowledge” for the prediction phase.

In the classification setting, the KNN algorithm essentially boils down to forming a majority vote between the K most similar instances to a given “unseen” observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance given by

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + ... + (x_n - x'_n)^2} \tag{2.7}$$

but other measures, including the Manhattan, Chebyshev and Hamming distances [Mafteiu-Scai, 2013], can be more suitable for certain given settings.

More formally, given a positive integer K, an unseen observation x and a similarity metric d, KNN classifier performs the following two steps:

- It runs through the whole dataset computing **d** between **x** and each training observation. We'll call the **K** points in the training data that are closest to **x** the set $\mathcal{A}$.

Note that K is usually odd to prevent tie situations.

- It then estimates the conditional probability for each class, that is, the fraction of points in A with that given class label. (Note $\mathbf{I}(\mathbf{x})$ is the indicator function which evaluates to 1 when the argument x is true and 0 otherwise)

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j) \tag{2.8}$$

Finally, the input x gets assigned to the class with the largest probability.

KNN was used to discriminate GPCRs from non-GPCRs and subsequently classify GPCRs at four levels on the basis of amino acid composition and dipeptide composition of proteins [Gao et al., 2006]. The prediction performance was evaluated on a non-redundant dataset consisted of 1406 GPCRs for six families and 1406 globular proteins using the Jackknife test. A high overall accuracy has been achieved in each step using dipeptide-based method. Moreover, comparisons with existing methods in the literature show that their method achieves great competitive performance.

Classification tree

Tree-based methods for regression and classification [James et al., 2013] involve stratifying or segmenting the predictor space into a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs. There are several tree construction algorithms, such as CART classification and regression Tree) , OC1, ID3 and C4.5 [Breiman et al., 1984; Quinlan, 1993; Salzberg et al., 1998]. An advantage of the tree-based classification algorithm is its relative ease of interpretation, which can help the user to understand and improve the classification rules.

Classification tree is a useful qualitative-response prediction algorithm that has been used in many areas of bioinformatics, such as gene finding, tumor classification, etc. [Salzberg et al., 1998; Zhang et al., 2001; Dudoit et al., 2002]. We predict that each observation belongs to the most commonly occurring class of training observation in the region to which it belongs.

In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region. Since we plan classification to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate E is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{2.9}$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.
The Gini index, which is a measure of total variance across the K classes, is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk} \tag{2.10}$$

There is also the cross entropy index [Kingsford et al., 2008]. Both the Gini index and the cross-entropy will take on a small value if the mth node is pure. Either of these indices can be used to evaluate the quality of a particular split, since they are more sensitive to node purity than is the classification error rate. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

The use of tree based algorithm for prediction and classification is very effective and is used to address several classification problems. Huang et al. (2004) described a bagging classification tree [Breiman, 1996] for classifying GPCRs into subfamilies and sub-subfamilies based on the amino-acid composition. The C4.5 algorithm was used to generate the 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.

The C4.5 algorithm uses a divide-and-conquer approach to growing decision trees. This algorithm selects a feature (amino acid composition) to split the training data into subsets (nodes of the decision tree). The default criterion used by C4.5 for feature selection is 'information gain ratio', a measure based on information theory [Shannon, 1948]. This measure can quantify how well a given feature separates the training data. At each node the training dataset will be further divided until some stopping criteria are satisfied. Then each terminal subset (leaf node) is assigned to a class label (receptor type). After generating the maximal classification tree, a pruning technique is used to simplify the classification tree and avoid over-fitting. Pruning a tree consists of replacing a whole sub-tree by a leaf node. The replacement takes place if the expected error rate in the subtree is greater than that in the single leaf. Aside from GPCR classification, the C4.5 algorithm has also been applied to genomic sequence annotation and protein phosphorylation sites identification [Kretschmann et al., 2001; Muggleton et al., 2001; Berry et al., 2004].

2.3.2 Unsupervised machine learning algorithms

Principal component analysis (PCA)

PCA refers to the process by which principal components are computed, and the subsequent use of these components in understanding the data [Jolliffe, 1989]. PCA is an unsupervised approach since it involves only a set of features $X_1, X_2, ..., X_p$ and no associated response Y. PCA finds a low-dimensional representation of a data set that contains as much as possible of the variation. Each of the dimensions found by PCA is a linear combination of the p features. Principal components (PCs) allow us to summarize large set of correlated variables with a smaller number of representative variables that collectively explain most of the variability. PCs are directions in feature space along which original data are highly variable and also define lines and subspaces that are as close as possible to the data cloud.

The first principal component of a set of features $X_1, X_2, ..., X_p$ is the normalized linear combination of the features.

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p \tag{2.11}$$

that has the largest variance. By normalized, we mean that $\sum_{j=1}^{p} \phi_{ji}^2 = 1$ where the $\phi_{11}, ..., \phi_{p1}$ are the loadings of the first principal components.

Given a $n \times p$ data set $\mathbf{X}$, how do we compute the first PC? Since we are only interested in variance, we assume that each of the variables in $\mathbf{X}$ has been centered to have mean zero (that is the column means of $\mathbf{X}$ are zero). We then look for the linear combinations of the sample feature values of the form ;

$$z_{i1} = \phi_{11}X_{i1} + \phi_{21}X_{i2} + ... + \phi_{p1}X_{ip} \tag{2.12}$$

that has the largest sample variance, subject to the constraints that $\sum_{j=1}^{p} \phi_{ji}^2 = 1$

In other words, the first PC loading vector solves the optimization problem

$$\max_{\phi_{11},...,\phi_{p1}} \frac{1}{n} \sum_{i=1}^{n} z_{i1}^2 \tag{2.13}$$

subject to

$$\sum_{j=1}^{p} \phi_{ji}^2 = 1 \tag{2.14}$$

Since $\frac{1}{n}\sum_{i=1}^{n} x_{ij} = 0$, the average of $z_{11}, ..., z_{n1}$ will be zero as well. Hence the objective of maximizing (2.13) is just the sample variance of n values of z_{i1}.

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 on 1497 sequence derived features which was further reduced into 32-dimensional feature vectors using the PCA. By the first level classifier a new protein sequence is predicted to be either a GPCR or a non-GPCR. If it is predicted to be a GPCR, it will be further classified into one of the four families, which is done by the second-level classifier.

The third-level classifiers hence determine which subfamily the protein belongs to. For some subfamilies, the fourth-level classifiers are used to determine the sub-subfamily of the protein. Finally, the fifth-level classifiers determine the subtypes of the protein, if any. Jackknife tests showed that the overall accuracies of PCA-GPCR are 99.5%, 88.8%, 80.47%, 80.3%, and 92.34% for the five levels, respectively.

2.3.3 Supervised statistical methods

Logistic regression

A logistic regression model [Hastie et al., 2009] is defined as:

$$h(x) = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n \tag{2.15}$$

where $x_1, x_2, x_3..., x_n$ are input features whereas $w_1, w_2, w_3, ..., w_n$ denote the regression coefficients.

Thus, the probability of the input to belong to a certain class can be defined as:

$$f(X) = (1 + e^{W^TX})^{-1} \tag{2.16}$$

where the variable X and W denote the vector of input features $[x_1, x_2, x_3, ..., x_n]$ and of the regression coefficients $[w_1, w_2, w_3, ..., w_n]$, also termed weights, respectively. Regularization is an essential technique in machine learning to counter over-fitting, which logistic regression implements in two forms: l_1 and l_2. Both have a λ parameter that is directly proportional to the penalty of finding complex or over-fitted models. The regularization term (i) in the l_1 form is the product of λ and the sum of the weights, while (ii) in the l_2 form (used here) it is the product of λ and the sum of the squares of the weights. The target value is expected to be a linear combination of the features considered.

Logistic regression has not been used for GPCR protein sequence prediction and classification which led us to do extensive research on its performance. In this work, I have developed a logistic regression as a proposed method for protein sequence prediction that will be described later in subsequent chapter. While not directly related to the GPCR classification problem, it should be mentioned that logistic regression was used to predict GPCR coupling [Singh G. et al., 2019], which goes further to; (i) predict GPCRs coupling probabilities to individual G-proteins; (ii) visually analyze the structural features and protein sequence that accounts for a particular coupling; and finally to (iii) suggest mutations based on predetermined coupling features to rationally design artificial GPCRs with new coupling properties . A tool was developed from this called PRECOG which was built by using experimental binding affinities of 144 human Class A GPCRs for 11 chimeric G-proteins obtained through TGFα shedding assay. From their work, a set of sequence and structure-based features were also derived that were statistically associated with each of 11 G-proteins, and could be used to devise predictive models.

Family specific motif

Due to the significant roles of GPCR acting both as receptors for outside ligands (ligands range from photons inducing sight to small peptides inducing neurological effects) and actuators for internal processes, it is very important to be able to distinguish which ligands that a specific GPCR interacts with and which parts of the sequence have a particularly important role. As a result, there were two presiding goals for computational methods in GPCR research: to classify GPCR sequences with respect to subfamilies within Class A which contains more than 80 percent of human GPCRs, and to identify the key ligand-interacting sites using the sequence alone.

In response to the above needs, Cobanoglu et al., (2011) developed a classification technique that also pinpoints ligand-receptor interaction sites. Their pro- posed technique involved identifying the frequent residue triplets in the sequence using sequence-derived motifs, calculating their distinguishing power among the subfamilies by the Distinguishing Power Evaluation (DPE) technique, and deducing rules from this information.

The classifier proposed here was called GPCRBind. It is a rule extraction method, and while training takes time on the order of hours, classification of a sequence takes milliseconds. This property of GPCRBind makes it suitable for being used as a classification server. The GPCRBind method, requires random partitioning. Due to this randomness the results of two successive runs are not identical. Therefore, they repeated the whole method 100 times and reported the average accuracy. They implemented the proposed methods and tested them on real data sets. The results were compared with the state-of-the-art GPCR classification techniques. Experiments showed that GPCRBind outperformed the state-of-the-art classification techniques.

Binary topology pattern (BTP)

In a paper by Inoue et al., (2004), they proposed a novel and simple (i.e., fast) method for the functional classification and identification of mammalian-type GPCRs based on the TM topology patterns [Poluliakh et al., 2000; Sugiyama et al., 2003], instead of using techniques based on functional domain detection or sequence similarity searches.

In their method, each loop length is expressed as '1' for a long loop or '0' for a short loop, and the TM topology pattern was expressed as a string of binarized loop lengths. The BTP method was expected to even identify and classify GPCRs with low amino acid sequence similarities, that sequence similarity searches such as BLAST [Altschul et al., 1990] cannot easily identify or classify, e.g., searching for mammalian-type GPCRs in invertebrate organisms. Comprehensive functional classification and identification of mammalian-type GPCRs were carried out by applying the BTP method to 10 different eukaryotic species

(H. sapiens, M. musculus, F. rubripes, C. intestinalis, A. thaliana, D. melanogaster, A. gambiae, C. elegans, P. falciparum, S. cerevisiae) for which the complete genome sequences or draft sequences have already been released.

BTP is 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 more than 80% for four groups.

Statistical encoding method

In a work by Iqbal et al., (2016), a statistical distance-based encoding method is used to represent a variable length lower similarity protein sequence with a fixed size vector. The protein sequences of three GPCRs families were investigated in the experiments due to their importance in the pharmaceutical industry. The sequences belong to the families of GPCRs having very weak sequence similarity which exhibits difficulties in classifying them into different classes. Each phase of their proposed framework was employed to perform a specific task of classifying protein sequences into their corresponding families/subfamilies.

The classification results obtained were the average results of various rounds of the classifier. Different performance measurement metrics such as accuracy, true positive rate (TPR), false positive rate (FPR), specificity, sensitivity, recall, F-measure and Mathews Correlation Coefficient (MCC) were investigated for performance evaluation of the protein sequence classification technique. The best accuracy results on Datasets 1 and 2 were in the range of 94% to 97.9%, which shows some improvement from the accuracy results which were in the range of 90.7% to 95% in the previous studies [Zhou et al., 2010 and Cobanoglu et al., 2011].

Structural regional lengths

In the study by Sahin et al. (2014), they proposed a method named GPCRsort, which determines the class of a GPCR using its structural properties. Specifically, they used the lengths of the transmembrane and loop regions of the GPCR structure. The method first determined the transmembrane regions of GPCRs, and constructs feature vectors using the lengths of the regions.

Random Forest [Breiman, 2001] classifier is employed in the learning and decision parts of the method. The method can predict GPCRs at every level of the GPCR class hierarchy tree. This shows the generality of GPCRsort, as the other techniques are inadequate to make the exact classification. The experimental results show that GPCRsort is very effective and outperforms the then known techniques for GPCR classification. Highest accuracy, 97.3%, is achieved with GPCRsort method when compared to the other techniques under the same experimental conditions. Besides these advantages, GPCRsort was shown to be a faster prediction algorithm among others.

2.3.4 Combination of tools

Genetic ensemble

Naveed and Khan (2012) introduced GPCR-MPredictor which predicts and classifies GPCRs into five levels (family, subfamily, sub-subfamily, subtype) including the prediction. In order to handle the dimesionality problem of the dipeptide compositions, they used genetic algorithm (GA)-based feature selection to reduce the dimensionality and to improve the classification accuracy.

Their approach is an ensemble approach where individual classifiers like KNN, SVM, probabilistic neural networks, J48, Adaboost and Naive Bayes classifiers have been used. Also, different thresholds in CD-Hit is used on the datasets downloaded from GPCRdb, that is 0.4, 0.7, 0.8, and 0.9 for the family, subfamily, sub-subfamily, and subtype levels respectively. However, they observed that SVM performs better on dipeptide composition among all of these classifiers. Amino acid composition, pseudo amino acid composition and dipeptide composition are the three features used to predict and classify GPCRs.

The proposed method is compared with the PCA-GPCR approach (Peng et al., 2010). The predictive performance of their approach is comparatively higher than that of PCA-GPCR at all levels of GPCRs. The GPCR-MPredictor yields an accuracy of 99.75%, 92.45%, 87.80%, 83.57% and 96.17% at the five levels, respectively. Their research shows that the smaller the number of training sequences, the less reliable the classifier might be which is evident in the decline in accuracies in levels 2, 3 and 4 but since a less strict CD-HIT threshold was applied in fifth level it results in a larger number of training sequences.

Fast Fourier transform with SVM

Guo et al. (2006) introduced a Fast Fourier transform based SVM method to classify GPCRs and Nuclear Receptors (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 Jackknife test was used for performance measurements as well as for prediction quality, accuracy, total accuracy and Matthew's correlation coefficient were evaluated. Higher accuracy is achieved with the hydrophobicity scale than c-p-v or electron-ion interaction potential model.

SVM with different approaches

Yabuki et al. (2005) have 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, Olfactory subfamilies of Class A, Class B, Class C, frizzled and smoothened) HMM is used for predicting the coupling selectivity with G-proteins which is outside the scope of my work. 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.

Karchin et al. (2002) have used a simple nearest neighbor approach like BLAST, a hidden Markov model (HMM) and SVMs to constitutue 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.

Li et al. (2010) proposed a three-layer classifier 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. In every level, minimum redundancy maximum relevance (mRMR) [Peng et al., 2005] is utilized to pre-evaluate features with discriminative information. After that, to further improve the prediction accuracy and to obtain the most important features, genetic algorithms (GA) [JH 2005] is applied to feature selection.

Finally, three models based on SVM are constructed and used to identify whether a query protein is GPCR and which family or subfamily the protein belongs to. The prediction quality evaluated on a non-redundant dataset by the jackknife cross-validation test exhibited significant improvement compared with published results. Higher accuracy is observed with the proposed method named GPCR-SVMFS than GPCR-CA and GPCRPred.

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%.

Chapter 3

Materials and Methods

In this chapter we describe the construction of the dataset that includes GPCRs and non-GPCRs. This is followed by a description of the features extracted from these protein sequences for predicting and classifying the GPCRs and how the features have been analyzed. The last section is the description of the algorithms used.

3.1 Data collection

For training and evaluating the GPCR prediction and classification algorithms, I constructed a non-redundant dataset consisting of full-length GPCRs that had been categorized to their subtype level by GPCRdb as well as all the non-GPCR protein sequences queried from the GPCR Prediction Ensemble database (GPCR-PEnDB) developed by our research team [Begum et al., 2020] and accessible at gpcr.utep.edu. The non-GPCRs in my dataset included both transmembrane and non-transmembrane sequences that are not GPCRs. These were selected from Uniprot with a CDHIT of 90%. A CDHIT of x% essentially produces a subset of protein sequences that are no more than x% similar to one another. The entire data collection contains 2016 positive examples of confirmed GPCRs, and 3100 non-GPCRs of which 1898 are transmembrane proteins. All these sequences were saved in a FASTA format file.

3.2 Feature extraction

In order to predict and classify GPCRs, different sets of features are used by different algorithms to measure the accuracies of the methods as well as the importance of those features for increasing performance. It is therefore imperative to have the capability of extracting a large variety of features from each (sequence) in the dataset. The objective here is to identify distinctive features used in different studies and later generate them uniformly for all sequences in our dataset using the programming language Python. A Python 3 script was written (see Appendix A1, A2) to generate a CSV (comma separated values) file that contains the sequence IDs and feature values.

The sequence-derived structural and physicochemical properties, such as hydrophobicity scales, average flexibility indices, polarizability parameter, etc., are highly useful for representing and distinguishing proteins or peptides of different structural, functional and interaction profiles that are essential for the successful application of statistical learning methods irrespective of sequence similarity [Han et al., 2004]. Recently, these structural and physicochemical features of proteins and peptides are routinely used in chemogenomics studies to characterize target proteins in therapeutic drug–target pairs and predict new drug–target associations to identify potential drug targets [He et al., 2010].

Several programs for computing protein structural and physicochemical features have been developed [Du et al., 2012; Holland et al., 2008; Li et al., 2006]. However, they are not comprehensive and not easily accessible.

Cao et al. (2013) published a selection of sophisticated protein features and provided them as a package called propy for the free and open source software environment Python. The original propy package was implemented in python 2 but I converted it to Python 3. The usefulness of the features covered by propy for computing structural and physicochemical features of proteins and peptides was already tested by a number of published studies (e.g., see Peng et al., 2010, Karchin et al., 2002, Bock et al., 2001), which reported prediction performance with sensitivity and specificity in the range of 82.1–99.9%. Because of the use of these structural and physicochemical features, these statistical and machine learning classification systems do not rely on sequence similarity, clustering or profiles for predicting protein functional classes, and they have been found to be particularly useful for facilitating the prediction of novel proteins.

In this dissertation, I used propy to compute 1360 protein features, grouped into four sets of descriptors with greater details in the subsections below, based on the structural and physicochemical properties listed in Table 3.1. Numerical values reflecting the properties of the amino acids were obtained from the physicochemical numerical indices in AAindex database [Kawashima et al., 2000]. These numerical values help capture as much information of the protein sequences as possible and they were used to compute the feature values for each protein in our dataset. The range of values for all the physicochemical properties are also shown in Table 3.1.

Table 3.1: The physicochemical properties of the amino acids

Order	Physicochemical property	Range of property
1	Hydrophobicity scales	[-1.14, 1.81]
2	Average flexibility indices	[0.295, 0.544]
3	Polarizability parameter	[0, 0.409]
4	Free energy of solution in water	[-2.24, 4.91]
5	Residue accessible surface area in tripeptide	[75, 255]
6	Residue volume	[36.3, 135.4]
7	Steric parameter	[0, 1.02]
8	Relative mutability	[18, 134]

Propy package description

The propy package [Dong-Sheng et al., 2013] is one of the tools used for computing commonly-used structural and physicochemical features of proteins and peptides from amino acid sequence [Dong-Sheng et al., 2013]. From the propy package, I used four groups of descriptors, each of which has been independently predicting protein and peptide-related problems by using machine-learning methods.

I wrote python scripts that call the various propy modules to extract required descriptors either by providing a fasta file of sequences (offline script, see Appendix A2) or a csv file that contains the protein ids for the sequences to be downloaded from the uniprot website (online script, see Appendix A1). The online script has the disadvantage of being interrupted by bad internet connection so it's always better to manually download the protein sequences in fasta format and use the offline script (see Appendix A2) to generate the descriptors. Both scripts skips sequences with special characters (X, B, Z, J, ϕ, ω, ψ, π, ζ, +, -). So manually downloading the fasta file helps in removing those characters before using the propy.

3.2.1 Amino acid and dipeptide compositions

Amino acid composition (AAC) and dipeptide composition (DC) are the simplest descriptors of protein sequence-derived features [Shepherd et al., 2003], that have been used for predicting protein fold [Grassmann et al., 1999] and structural classes [Reczko et al., 1994], functional classes [Bhasin et al., 2004] and subcellular locations [Hua et al., 2001] at accuracy levels of 72–95%, 83–97% and 79–91%, respectively.

AAC is defined as the occurrence frequencies of 20 amino acids in a protein sequence. That is,

$$f_A(i) = \frac{n_A(i)}{L}, \tag{3.1}$$

where each $i = 1, 2, ..., 20$ corresponds to a distinct amino acid and $n_A(i)$ is the number of amino acid i occurring in the protein sequence of length L. AAC was widely used to transform GPCR sequences into 20-dimension numerical vectors [Chou et al., 1999]. However, the sequence order information would be completely lost. In order to address this issue, DC was proposed to represent GPCR sequences by 400-dimension vectors, which capture local-order information and have been reported to improve classifications.

Similarly, DC [Bhasin et al., 2004] is defined as the occurrence frequencies of the 400 dipeptides (i.e., 400 amino acid pairs). That is,

$$f_D(i) = \frac{n_D(i)}{L-1}, \tag{3.2}$$

where each $i = 1, 2, ..., 400$ corresponds to one of the 400 dipeptides and $n_D(i)$ is the number of dipeptide i occurring in the sequence.

3.2.2 Autocorrelation descriptors

Autocorrelation descriptors (ADs) are a class of topological descriptors, also known as molecular connectivity indices. They describe the level of correlation between two objects (protein or peptide sequences) in terms of their specific structural or physicochemical property [Broto et al., 1984], which are defined based on the distribution of amino acid properties along the sequence.

There are three ADs – normalized Moreau-Broto autocorrelation descriptor, Moran autocorrelation descriptor and Geary autocorrelation descriptor. They are all defined based on the value distributions of the eight physicochemical properties of amino acids (see Table 3.1) along a protein sequence. The measurement values of these properties are first centralized and standardized before proceeding to calculate the autocorrelation descriptor. The standardization is performed as follows;

$$P(i) = \frac{P_0(i) - \bar{P}_0}{\sigma} \tag{3.3}$$

where $P_0(i)$ are the property value of the amino acid i,

$$\bar{P}_0 = \frac{1}{20}\sum_{i=1}^{20} P_0(i), \qquad \sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(P_0(i) - \bar{P}_0)^2} \tag{3.4}$$

The normalized Moreau-Broto autocorrelation (NMBA) descriptor is defined as:

$$NMBA(d) = \frac{MBA(d)}{L-d} \qquad d = 1, 2, 3, ..., 30 \tag{3.5}$$

where

$$MBA(d) = \sum_{i=1}^{L-d} P(R_i)P(R_{i+d}), \tag{3.6}$$

R_i and R_{i+d} are the amino acids at position i and $i+d$ along the protein sequence, respectively. NMBA has been used for predicting transmembrane protein types [Feng et al., 2000] and protein secondary structural contents [Lin et al., 2001] at accuracy levels of 82–94% and 91–94%, respectively.

The Moran autocorrelation (MA) descriptor [Moran 1950] is defined as:

$$MA(d) = \frac{\frac{1}{L-d}\sum_{i=1}^{L-d}(P(R_i) - \bar{P})(P(R_{i+d}) - \bar{P})}{\frac{1}{L}\sum_{i=1}^{L}(P(R_i) - \bar{P})^2} \qquad d = 1, 2, 3, ..., 30 \tag{3.7}$$

where $\bar{P}_0 = \frac{1}{L}\sum_{i=1}^{L} P(R_i)$ is the average value of the property of interest along the sequence. This has been applied for predicting protein helix contents at an accuracy level of 85% [Horne et al., 1988].

The Geary autocorrelation (GA) descriptor [Geary, 1954] which has been used for analyzing allele frequencies and population structures [Sokal et al., 2006] is defined as:

$$GA(d) = \frac{\frac{1}{2(L-d)}\sum_{i=1}^{L-d}(P(R_i) - P(R_{i+d}))^2}{\frac{1}{L-1}\sum_{i=1}^{L}(P(R_i) - \bar{P})^2} \qquad d = 1, 2, 3, ..., 30 \tag{3.8}$$

MBA uses the property values as the basis for measurement, MA utilizes property deviations from the average values, and GA utilizes the square-difference of property values instead of vector-products (of property values or deviations). The MA and GA descriptors measure spatial autocorrelation, which is the correlation of a variable with itself through space [Ong et al., 2007].

3.2.3 Pseudo-amino acid composition

The pseudo-amino acid composition (PseAAC) descriptor contain a set of sequence features originally developed by Chou K.C. (2001) . They have been used widely to predict outer membrane protein [Cai et al., 2003], nuclear receptors [Gao et al., 2006], and protein structural classes [Xiao et al., 2005]. PseAAC helps to avoid completely losing the sequence-order information [Chou, 2001, 2005] in calculating the AAC and is a good indicator for subcellular localizations [Chou 2000, Shengli et al., 2017] of protein.

Given a protein sequence P with L amino acid residues, i.e.,
$P = [R_1, R_2, R_3, ..., R_L]$, which can then be formulated as
$P = [p_1, p_2, p_3, ..., p_{20}, p_{20+1}, ..., p_{20+\lambda}]^T$, $(\lambda < L)$. Here we take λ to be 30.
Finally, the Chou's pseudo amino acid composition descriptor is defined as:

$$PseAAC(i) = \begin{cases} \frac{f_A(i)}{\sum_{j=1}^{20} f_A(i)+\omega\sum_{d=1}^{30}\theta(d)} & 1 \le i \le 20 \\ \frac{\omega\theta(i)}{\sum_{j=1}^{20} f_A(i)+\sum_{d=1}^{30}\theta(d)} & 21 \le i \le 50 \end{cases} \tag{3.9}$$

where ω is a weighting factor (default $\omega = 0.1$) and $\theta(d)$ the dth-tier correlation factor that reflects the sequence order correlation between all the dth most contiguous residues as formulated by

$$\theta(d) = \frac{1}{L-d}\sum_{i=1}^{L-d}\Theta(R_i, R_{i+d}), \qquad d = 1, 2, ..., 30 \tag{3.10}$$

where $\Theta(R_i, R_{i+d})$ is the dth-tier correlation factor that reflects the sequence order correlation between all the most contiguous residues along a protein chain. It is defined as:

$$\Theta(R_i, R_{i+d}) = \frac{1}{3}\sum_{k=1}^{3}[H_k(R_i) - H_k(R_{i+d})]^2 \qquad (3.11)$$

where $H_1(R_i)$, $H_2(R_i)$ and $H_3(R_i)$ are the hydrophobicity, hydrophilicity, and side-chain mass of amino acid, respectively [Chou, 2001]. Their original values are standardized.

3.2.4 Sequence-order descriptors

The sequence-order (SD) descriptors include sequence-order coupling numbers and quasi sequence-order. They have been used for predicting protein subcellular locations at accuracy levels of 72.5–88.9% [Chou, 2000]. In deriving the sequence-order descriptors, we depend on two distance measures for amino acid pairs which are the Grantham chemical distance matrix [Grantham, et al., 1974], and the other called Schneider-Wrede physicochemical distance matrix [Chou, KC 2000, Schneider et al., 1994].

Then, the jth-rank sequence-order coupling number is defined as:

$$\tau(j) = \sum_{i=1}^{L-j}(d(R_i, R_{i+j}))^2, \qquad j = 1, 2, ..., 30 \qquad (3.12)$$

where $d(R_i, R_{i+j})$ is one of the above distances between the two amino acids R_i and R_{i+j} located at position i and $i + j$, respectively.

The quasi-sequence-order descriptors are defined as:

$$QSO(i) = \begin{cases} \frac{f_A(i)}{\sum_{j=1}^{20} f_A(j)+\omega\sum_{j=1}^{30}\tau(j)} & 1 \leq i \leq 20 \\ \frac{\omega\tau(i)}{\sum_{j=1}^{20} f_A(j)+\sum_{j=1}^{30}\tau(j)} & 21 \leq i \leq 50 \end{cases} \qquad (3.13)$$

where ω is a weighting factor (default $\omega = 0.1$). We end up with 90 sequence-order-coupling numbers and 100 quasi-sequence-order descriptor values. In total, there are 190 features extracted from the sequence-order descriptors.

3.3 Methods

In this section, the various methods applied to obtain the sequence-derived features are outline. The first part is the feature selection method followed by how the imbalance nature of the data in some levels were handled. Then we look into the classifiers, and finally consider the performance metrics.

3.3.1 Feature selection

There is this saying by Mills (1993) about data that goes, if you torture your data long enough, they will tell you whatever you want to hear. In contrast to other different dimension reduction techniques like those based on projection (e.g., principal component analysis) or compression (e.g., using information theory), feature selection (FS) do not modify the original representation of the variables, but rather select a subset of them [Song et al., 2017]. Thus, the original semantics of the variables are preserved thereby providing the advantage of easy interpretation by an expert in the domain.

The use of all available descriptors likely results in the inclusion of partially redundant information, some of which might become noise that interferes with the prediction results or obscures relevant information. Based on the results of previous studies, it is possible that applying FS methods for selecting the optimal set of descriptors will help to improve prediction accuracy as well as computing efficiency.

Importantly FS allows us to avoid overfitting and improve prediction performance, provide faster and more cost-effective models, and gain a deeper insight into the underlying processes that generated the data. I have surveyed various feature selection methods like f-score [Chen et al., 2003] and FDA-score [Song et al., 2017] but finally settled with the univariate filter function known as f-classif because its selected features produced better results. This method assesses the relevance of the features by looking only at the intrinsic properties of the data by implementing the ANOVA test on each feature. A feature relevance score and the low scoring features are removed (that is the best ones are selected). The f-classif function is in scikit-learn library [Pedregosa et al., 2011] and it works by computing the ANOVA F-value for the provided sample. Scikit-learn library in python (also known as sklearn) is an algorithmic library with machine learning tools built on two scientific computing libraries (NumPy and SciPy) and one visualization library (Matplotlib). f-classif takes two arguments which are the feature sets and the response variable.

3.3.2 Handling imbalance data

An imbalanced dataset is when the classes are approximately unequally represented. In the superfamily level, there are 1084 more negative examples than the positive examples in our dataset. Similarly, in the family level, Class A is atleast six times as many as the T2R (second largest class within this family). In order to deal with such data imbalance, we introduced the Synthetic Minority Oversampling Technique (SMOTE) [Chawla et al., 2002] to be used with any of the classifiers on levels 1 [Liao et al., 2016] and 2 of which the proposed ones were used. With SMOTE, the minority class in the training set is over-sampled by creating "synthetic" examples. Its detailed operation is as follows:

For each sample x in the minority class, search its k (usual value 5) nearest neighbors, if the sampling rate is N, then randomly select N samples without replacement from the k nearest neighbors, denoted as y_1, y_2, . . . , y_N. Random linear interpolation is performed on the line segments formed by x and y_i $(i = 1, 2, ..., N)$ to obtain a new positive sample z_{new}, as shown in the following equation:

$z_{new} = x +$ rand [0, 1] $\times$ $(y_i - x)$

where rand(0,1) means to generate a random number between 0 and 1. Add these newly synthesized samples to the original positive sample and form a new, more balanced dataset. After performing SMOTE, we can have a more balanced dataset, where minority and majority examples are similar in numbers, to be fed into the classifier.

In python, the class imblearn.over_sampling contains the SMOTE function with several parameters like sampling_strategy, random_state, k_neighbors etc. The sampling_strategy can be “auto” equivalent to not majority, “all” equivalent to resampling all classes but the majority class, “not minority” equivalent to resampling all classes but the minority class and “minority” equivalent to resampling only the minority class. The k_neighbors can be any integer preferably odd number. In this work, SMOTE was applied in the family level and the default parameters were maintained (sampling_strategy=‘auto’, k_neighbors= 5).

3.3.3 Classifiers

Four different classifiers were considered, the first two are established algorithms already used for GPCR sequence classification [Peng et al., 2010, Bhasin & Raghava 2004] and the last two are new methods developed in my research. I will describe all four methods here and their performance of these methods will be compared using the performance measures presented later in Section 3.4.

- Principal component analysis & intimate sorting (PCA-ISA)
- Support vector machine (SVM)
- Penalized multinomial logistic regression (Log-Reg)
- Multi-layer perceptron neural network (MLP-NN)

SVM, Log-Reg and MLP-NN were implemented with scikit-learn library to build models using the training set which has the same 1360 features but different rows across the levels. The parameters that were implemented within these classifiers were fine tuned and the optimal ones selected.

PCA-ISA

PCA [Jolliffe, 1989] is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components (PCs). PCA finds a low-dimensional representation of a dataset that contains as much as possible of the variation. Each of the dimensions found by PCA is a linear combination of the all features. Peng et al. (2010) used PCA to reduce the dimension of their GPCR data and then apply the intimate sorting algorithm (ISA) [Chou et al. 1999] as the classifier used after retrieving the PCs. ISA algorithm is easy to implement and does not need to set any parameters as some other algorithms (e.g., support vector machines).

Suppose that a training set consists of N proteins P_1, P_2, ..., P_N, each of which P_i is a λ-dimension vector, $\mathbf{P_i}$= $(p_{i,1},\ p_{i,2},\ .\ .\ .,\ p_{i,\lambda})^T$. The classes of these proteins are already known, and each protein belongs to exactly one of the μ classes. The intimate sorting algorithm aims to place a query protein $\mathbf{P}$= $(p_1,\ p_2,\ .\ .\ .,\ p_\lambda)^T$ into one of the μ classes based on the information from the N proteins in the training set.

To this end, a measure of similarity score between P and P_i is defined as

$$\Phi(P, P_i) = \frac{P \cdot P_i}{||P||||P_i||}, \qquad\qquad i = 1, 2, ..., N$$

where $P{\cdot}P_i$= $\sum_{j=1}^{\lambda} p_j \cdot p_{i,j}$ and $||P||$= $\sqrt{\sum_{j=1}^{\lambda} p_j^2}$. When P $\equiv$ P_i, it can be easily seen that $\Phi(P, P_i)$= 1, meaning that they belong to the same class. In general, we have -1$\leq\Phi(P, P_i)\leq$1. The higher the $\Phi(P, P_i)$ value, the more likely two proteins belong to the same class. Among the N proteins in the training set, the one with the highest score with the query protein P is picked out, which we denote by $\mathbf{P_k}$, $k \in [1, N]$. If there is a tie, we would randomly select one of them. In the final step, the intimate sorting algorithm simply assigns P into the same GPCR class as $\mathbf{P_k}$.

In this work, PCA was implemented using the sklearn library. During the model preparation stage, where the data is partitioned as training and test, it works by accepting the minimum number between sample size and feature size as the maximum PCs required. A "ValueError" traceback results when PCs higher than the sample size is used as a parameter. And since all the levels have the same feature size (1360), it is the sample sizes that were changed across the levels meaning PCs cannot be more than the sample size. It was noticed after several trials that PCs that produced a total variation of between 80% and 97% resulted in optimal results. That is, the exact number of sample size (100% total variation) was not used. The resulting PCs were then fed into ISA which I implemented from scratch in python using NumPy library because the original Peng's code is not available. PCA-ISA code can be found in Appendix C.

SVM

An SVM [Vapnik, 1995] constructs a hyperplane or set of hyperplanes in a high dimensional space, which can be used for classification, regression or other tasks. We implemented the SVM in scikit-learn using the support vector classifier (SVC) module. Intuitively, a good separation is achieved by the hyperplane that has the largest distance (so-called functional margin) to the nearest training data points of any class. In general the larger the margin the lower the generalization error of the classifier.

For simplicity, imagine that a 1-dimensional dataset. This means we have one feature X. As it is not linearly separable we can transfer it into a 2-D space. For e.g, you can increase the dimension of the data by mapping x into a new space using a function with output x and x^2. We are now in 2-D space after the transformation from 1-D, the hyperplane is a line dividing a plane into two parts where each class lays on either side. The axes are the range of values of the two features.

Given training vectors $x_i \in i = 1, ..., n$ in two classes, and a vector $y \in \{-1, 1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\zeta}(\frac{1}{2}w^T w + C\sum_{i=1}^{n}\zeta_i) \tag{3.14}$$

subject to $y_i(w^T\phi(x_i) + \text{b}) \geq 1 - \zeta_i$,

$\zeta_i \geq 0$, $i = 1, ..., n$

It's dual

$$\min_{\alpha}(\frac{1}{2}\alpha^T Q\alpha - e^T\alpha)$$

subject to $y^T\alpha = 0$

$0 \leq \alpha_i \leq C \qquad\qquad i = 1, ..., n$

where e is the vector of all ones, C $>$ 0 is the upper bound, is an n by n positive semidefinite matrix.

$Q_{ij} = y_i y_j K(x_i, x_j)$ where $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is:

$$sgn(\sum_{i=1}^{n} y_i\alpha_i K(x_i, x) + \rho)$$

The advantages of SVM are:

• Accurate in high-dimensional space.

• Use a subset of training points in the decision function so it's also memory efficient.

The disadvantages are:

• It is prone to over-fitting if the number of features is much greater than the number of samples.

• Do not directly provide probability estimates, which are desirable in most classification problems.

In our work, after several testing of various parameters based on performance and efficiency, radial basis function (RBF) was used as the kernel and the stochastic gradient descent (SGD) as the optimizer. Also in terms of multi-class classification, the "one-against-one" approach for multi-class classification was adopted. That is, if n is the number of classes, then $n \times (n-1)/2$ classifiers are constructed and each one trains data from two classes. To provide a consistent interface with other classifiers, the "decision_function_shape" parameter allows to monotonically transform the results of the "one-against-one" classifiers to a decision function of shape (n_samples, n_classes).

Log-Reg

The basic logistic regression model [Hastie et al., 2009] for binary classification with m samples and n features is defined as:

$$h(x) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n \tag{3.15}$$

where $x_1, x_2, x_3, ..., x_n$ are input features whereas $\beta_1, \beta_2, \beta_3, ..., \beta_n$ denote the regression coefficients, also called weights. The prediction probability of a query is then taken to be

$$p(X) = P(Y = 1|X) = \phi(\beta^T X) = \frac{\exp(\beta^T X)}{1 + \exp(\beta^T X)} \tag{3.16}$$

where the X and β denote the vector of input features $[x_1, x_2, x_3, ..., x_n]$ and of the weights $[\beta_1, \beta_2, \beta_3, ..., \beta_n]$, respectively. The objective is to find a model that best estimates the actual labels. Finding the best model means finding the best weights for that model by minimizing the cost function through an iterative optimization approach.

With more than two classes, we use an extension of (3.16) called multinomial model, which, given K classes for the outcome Y, takes the form:

$$P(Y = 1|X) = \frac{\exp(\beta_1^T X)}{1 + \sum_{j=1}^{K-1} \exp(\beta_j^T X)} \tag{3.17}$$

$$P(Y = 2|X) = \frac{\exp(\beta_2^T X)}{1 + \sum_{j=1}^{K-1} \exp(\beta_j^T X)} \tag{3.18}$$

$$\vdots$$

$$P(Y = K-1|X) = \frac{\exp(\beta_{K-1}^T X)}{1 + \sum_{j=1}^{K-1} \exp(\beta_j^T X)} \tag{3.19}$$

$$P(Y = K|X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(\beta_j^T X)} \tag{3.20}$$

Regularization is an essential technique in machine learning to counter (or penalize) over-fitting, which can be implemented with a λ parameter that is directly proportional to the penalty of finding complex or over-fitted models. In a commonly used form of regularization, called l_2, the penalty term is the product of λ and the sum of squares of the weights. The target value is expected to be a linear combination of the features considered. The cost function represents the total error of the model which is the difference between the actual and the models predicted values.

Given a model $\hat{y}$ and actual label $y = 1$ or 0, a cost function of a single data point is defined as:

$$Cost(\hat{y}, y) = \frac{1}{2}(\phi(\beta^T X) - y)^2 \tag{3.21}$$

so the total cost function for the training set is:

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{m} Cost(\hat{y}, y) \tag{3.22}$$

In general, it is difficult to calculate the derivative of the cost function so -log of the model is used. That is, in a case that desirable $y = 1$, the cost can be calculated as -log($\hat{y}$)

and in the case that desirable $y = 0$, the cost can be calculated as -log$(1-\hat{y})$. Now plugging into (3.22), we get

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}) + (1 - y^i)\log(1 - \hat{y}) \tag{3.23}$$

which becomes the logistic regression cost function.

Now introducing the l_2 penalty and minimizing the results we then end up with:

$$\min_{\beta}(\frac{1}{2}\beta^T\beta + J(\beta))$$

In this work, the "LogisticRegression" function in scikit-learn library was used and the tuning was done on "multiclass", "penalty", "tol", and "solver" parameters. The best model was obtained when the parameters used were multinomial (one vs rest) multiclass, l_2 penalty and the lbfgs (limited broyden fletcher goldfab-shannon) solver as minimization technique with a tolerance (stopping criteria) of 0.01.

<u>**MLP-NN**</u>

MLP-NN, also available, in scikit-learn is a supervised learning algorithm that learns a function $f(.) : R^m \mapsto R^p$ by training on a dataset, where m is the number of dimensions for input and p is the number of dimensions for output.

Given a set of features $X = x_1, x_2, ..., x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, ..., x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function $g(.) : R \mapsto R$ - such as the logistic sigmoid function ("logistic"), no-op activation ("identity"), and hyperbolic *tangent* function ("tanh"). The output layer receives the values from the last hidden layer and transforms them into output values. The Class function *MLPClassifier* in scikit-learn library was used to implement the multi-layer perceptron neural network (MLP-NN) algorithm.

The advantage of using MLP-NN is its capability to learn non-linear models in real-time (on-line learning). The disadvantages are that hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy. It also requires tuning a number of hyperparameters such as the number of hidden layers, neurons and iterations. MLP-NN is sensitive to feature scaling.

Several combinations of parameters of the MLPClassifier were tuned to choose those that produce the best accuracy and efficiency. In the end, I found out one hidden layer with 50 neurons, the tanh as the activation function and lbfgs solver as the optimizer were parameters that gave the best results in terms of accuracy and efficiency. When more hidden layers are used, it takes too much time to produce results and might even lead to overfitting (unstable results).

3.4 Performance measures

To evaluate the generalization of the classification algorithms, all the data are partitioned randomly into 70% training and 30% test sets. The experiment is repeated 5 times and the average is taken which serves as the main classification accuracy used to assess the general performance. At each level of classification, there are different number of samples and classes but same set of features were used. The prediction accuracy (ACC) for each class and the overall classification accuracy (OACC) at a given level are then measured by the following formulas:

$$\text{ACC}(i) = \frac{C(i)}{Tot(i)} \times 100\%, \quad i = 1, 2, ..., \mu$$

$$\text{OACC} = \frac{\sum_i^\mu C(i)}{\sum_i^\mu Tot(i)} \times 100\%, \quad i = 1, 2, ..., \mu$$

where $Tot(i)$ is the total number of sequences in class i, $C(i)$ the number of correctly predicted sequences of class i, and μ the total number of classes at the level under consideration.

Table 3.2: Categories of superfamily classification results

		Predicted	
		GPCR	Non-GPCR
Actual	GPCR	True Positive (TP)	False Negative (FN)
	Non-GPCR	False Positive (FP)	True Negative (TN)

At the superfamily level, we have $\mu = 2$ with the GPCRs as class 1, and the non-GPCRs as class 2. In this setting, ACC(1) was the true positive rate (TPR) or sensitivity, which measured the fraction of true positives (actual GPCRs predicted correctly). ACC(2) was the true negatives rate (TNR) or specificity, which measured the fraction of true negatives (actual non-GPCRs predicted correctly). Refer to Table 3.2 for the meaning of the notations TP, FP, TN and FN in the superfamily level of GPCR classification.

As we are particularly interested to find out about the capabilities of the different algorithms in terms of distinguishing GPCRs from other non-GPCR transmembrane proteins, we also measured the false positive rates among transmembrane non-GPCRs (TmFPR) in addition to the overall false positive rates (FPR) among non-GPCRs, noting that FPR = 1 - TNR.

3.4.1 Accuracy assessment procedure

In assessing the accuracy, the data is randomly partitioned into different sets for training and testing within a for-loop, in which the algorithms are being run and the accuracy measured based on the results produced for each test set. At each step of the loop, the training set and test set were randomly chosen to be 70% and 30% of the data sequences respectively. The average overall accuracy across the 5 different randomly partitioned test sets is recorded.

3.5 Platform Used

All the algorithms used were tested and implemented on a Dell Inspiron 7586 laptop with Intel core i7-8565U CPU @ 1.80GHz and 16GB RAM. The time module in python was used to record the execution times.

Chapter 4

Results and Discussion

In this chapter, we deal with the datasets, features collected and results of the model performance of the proposed methods and other state-of-the-art algorithms.

4.1 Dataset

There are a total of 2016 GPCRs in my dataset and are distributed into their eight major classes: A (rhodopsin), B1 (adhesion), B2 (secretin), C (glutamate), D (fungal pheromone), E (cAMP receptor), F (Frizzled) and T2R (Taste2 receptors). More details of this classification of GPCRs can be found in [Begum et al. 2020] . They are highly dominated by class A which contains about 70% of all the GPCRs collected, followed by T2R with about 10%. The least represented is class E, constituting only 0.4%. The non-GPCRs are made up of two parts: the transmembrane (Tm) non-GPCRs and other proteins that are not Tm. Table 4.1 shows the numbers of proteins in our dataset separated into these categories. The raw data for GPCRs are stored in eight different fasta files according to their families where as all non-GPCRs in one fasta file.

Table 4.1: Number of collected proteins in different classes

Proteins	Family	Number of sequences
GPCR	Class A	1387
	Class B1	90
	Class B2	113
	Class C	112
	Class D	13
	Class E	11
	Class F	79
	Class T2R	211
Tm non-GPCRs		1898
Non-Tm non-GPCRs		1202
Total		5116

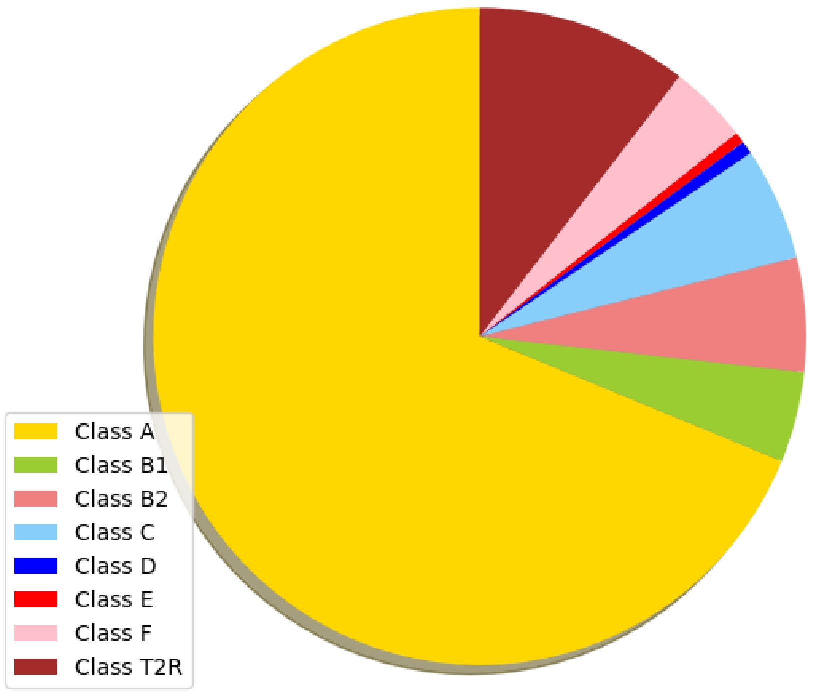


Figure 4.1: Distribution of GPCRs among eight different classes at the family level.

4.2 Features extracted and normalized

A list of 1360 features were computed using propy for each protein in our dataset. Table 4.2 displays the feature groups and their descriptors, along with number of features and range of values for the descriptors.

Table 4.2: Feature groups and number of descriptors

Feature groups	Descriptors	No. of features	Range of values
Sequence compositions	Amino acid composition (AAC)	20	[1 - 15%]
	Dipeptide composition (DPC)	400	[0 - 2%]
Pseudo amino acid	Pseudo amino acid (PseAAC)	30	[0.5 - 6%]
Sequence order	Sequence order coupling number (SOCN)	90	[200, 500]
	Quasi-sequence order descriptors (QSO)	100	[$1e^{-5}$ - 0.05]
Autocorrelation	Normalized Moreau-Broto autocorrelation (NMBA)	240	[-0.3 - 0.3]
	Moran autocorrelation (MA)	240	[-0.3 - 0.3]
	Geary autocorrelation (GA)	240	[0.1 - 1.2]

The processed data has the features of each sequence and GPCRs are stored in CSV formats categorized according to the family, subfamily, sub-subfamily and subtypes being considered. The non-GPCRs has only one CSV file. The full set of features were used to train the models for all the algorithms considered.

From Table 4.2, we notice that descriptors have different range of values and have to be made uniform and consistent by normalizing them. By making the ranges consistent between variables, normalization enables a fair comparison between different features making sure they have the same impact and also important for computational reasons. The standard_scaler function from scikit-learn.preprocessing module was used to standardize the columns by subtracting the mean and scaling to unit variance. The standard score of sample x is calculated as $z = (x-u)/s$ where u is the mean and s is the standard deviation.

4.3 Classification accuracies

The tables in this section are the results from all four algorithms for part of the GPCR classification cycle shown in Figure 4.2, which shows the details of each level of classification for class A (rhodopsin family). The entire collection of the classification tables are in Appendix Q. All the accuracy measurements reported are the average of 5 test runs using 30% of the entire data as testing sets.

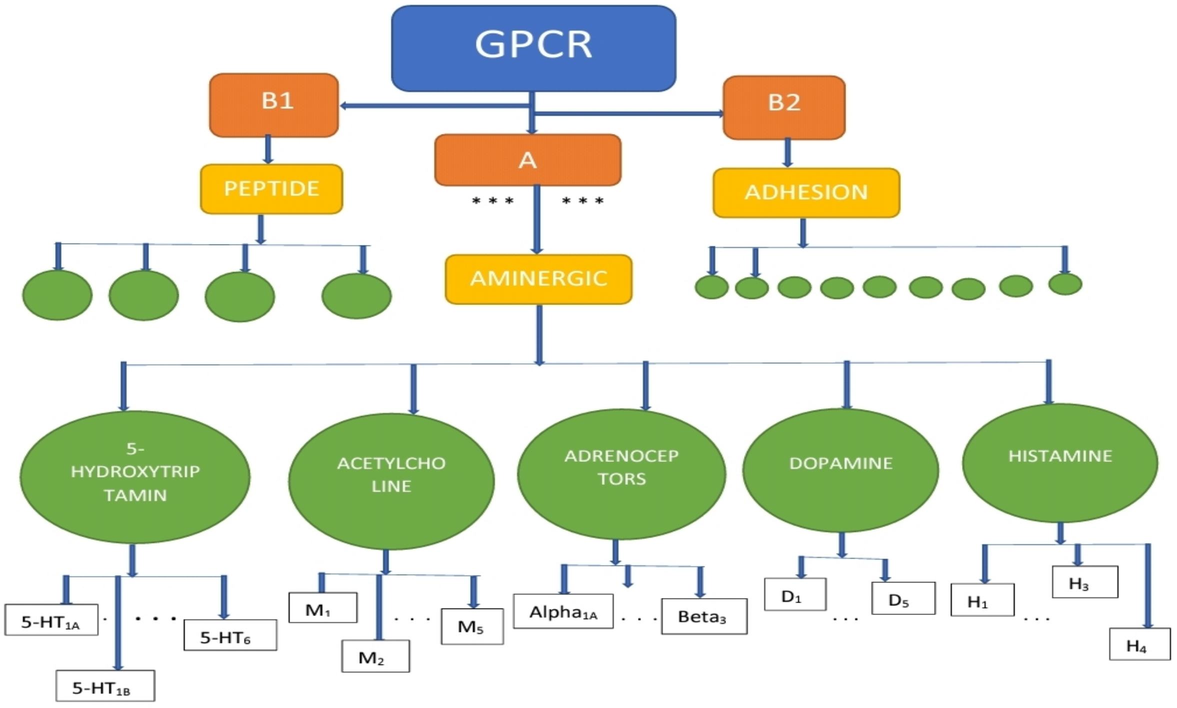


Figure 4.2: GPCR classification cycle to its subtypes level.

4.3.1 Classification at superfamily level

We begin with the superfamily (Level 1) classification, which is a binary classification process to distinguish GPCRs from non-GPCRs. All four classifiers (PCA-ISA, SVM, Log-Reg, MLP-NN) described in Section 3.3.3 are used. For each run, the train_test_split function in sklearn randomly splits the entire data into the specified testset size of 30% representing 1535 examples which includes both GPCRs and non-GPCRs. The average prediction accuracies ACC(1) and ACC(2), which are respectively equivalent to TPR and TNR as described in section 3.4, along with the overall prediction accuracy OACC are displayed in Table 4.3. While all classifiers gave excellent performance with OACC above 95%, it is particularly important to assess their capabilities of discriminating GPCRs from the structurally similar Tm non-GPCRs. The fourth column of numbers in Table 4.3 are the average percentages of Tm non-GPCRs that are mistakenly classified as GPCRs. Note that they are, as expected, uniformly higher than the FPRs for the non-Tm non-GPCRs (last column).

Table 4.3: ACC and OACC (both in %) of GPCR classification at superfamily level

	PCA-ISA	**SVM**	**Log-Reg**	**MLP**
TPR	97.6	98.5	98	98.5
TNR	94.5	98.0	97.0	98.5
OACC	95.4	97.8	97.2	98.1
FPR(Tm)	9.5	3.4	5.3	2.0
FPR(NTm)	2.3	0.7	1.2	0.3

With overall OACC over 98%, MLP-NN outperforms the rest of the classifiers in discriminating GPCRs from other proteins. Furthermore, it has the smallest overall type 1 error (FPR) of 1.5% with 2.0% and 0.3% among Tm and non-Tm non-GPCRs respectively. This means it has the largest chance of identifying Tm non-GPCRs as non- GPCRs.

4.3.2 Classification of GPCRs at family level

Table 4.4. gives the Level 2 average classification accuracies for each of the eight families of GPCRs together with the overall accuracies by the four classifiers. There are 605 GPCRs which have been classified under these families

Table 4.4: ACC and OACC (both in %) of GPCR classification at family level

Family	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
A	100.0	100.0	100.0	99.6
B1	97.6	99.0	100.0	99.2
B2	97.4	90.4	97.8	97.8
C	94.8	84.2	96.6	97.8
D	35.0	20.0	42.4	77.4
E	83.0	0	80.0	92.0
F	87.2	93.0	97.4	97.8
T2R	100.0	100.0	100.0	100.0
OACC	98.6	97.6	99.1	99.1

Log-Reg and MLP-NN outperform the rest with an OACC of 99.1% as against 98.6% and 97.6% by PCA-ISA and SVM respectively. Here we see class D, which contains only 13 GPCR examples, has the lowest ACC compared to other classes. However, MLP-NN still performs better than the rest. Class E also has few examples, only 11. We notice that SVM fails to classify any of the class E GPCRs correctly. Yet MLP-NN produced the best ACC of 92%. At this level, we experience the phenomenon of imbalance classes and one remedy to this is using SMOTE. In Table 4.22, we see the best performance when it is applied with Log-Reg.

4.3.3 Classification of GPCRs at subfamily level

Table 4.5 outlines the classification accuracies of the subfamilies (Level 3) under Class A, together with the overall accuracies, by all the classifiers. There are 417 testing examples in this family, that has subfamily classification information available.

Table 4.5: ACC and OACC (both in %) of GPCR classification of Class A family at subfamily level

Class A subfamilies	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
Alicarboxylic acid	96.0	43.0	100.0	100.0
Aminergic	100.0	98.2	100.0	99.8
Lipid	94.6	92.2	98.2	90.4
Melatonin	100.0	93.4	100	83.8
Nucleotide	96.8	93.2	96.8	92.0
Orphan	95.2	92.8	94.6	93.0
Peptide	96.8	98.8	98.0	97.8
Protein	100.0	98.4	99.4	98.6
Sensory	100.0	95.2	98.0	97.6
Steroid	100.0	59.0	99.2	100.0
OACC	97.3	95.8	97.9	96.3

We notice again from Table 4.5 that Log-Reg outperforms the rest with an OACC of 97.85% as against 97.33% and 95.75% by PCA-ISA and SVM respectively.

Class B1 with 90 sequences (both training and testing) has only one type of subfamily called Peptide, B2 also with one subfamily called Adhesion, C with four different subfamilies, F with one and finally D & E has none.

4.3.4 Classification of GPCRs at sub-subfamily level

Tables 4.6 - 4.10 give the classification accuracies of five Class A GPCR subfamilies (alicarboxylic acid, aminergic, lipid, nucleotide, and protein) into their sub-subfamilies (Level 4) by the four classifiers.

Table 4.6: ACC and OACC (both in %) of GPCR classification of Class A subfamily Alicaboxylic at sub-subfamily level

Alicaboxylic	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
Hydroxicarboxylic acid	100	100	100	100
Oxoglutarate receptors	100	100	100	100
Succinate receptors	100	100	100	100
OACC	100	100	100	100

Table 4.7: ACC and OACC (in %) of GPCR classification of class A subfamily Aminergic at sub-subfamily level

Aminergic	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
5-Hydroxytryptamine	98	95.4	96	99.2
Acetylcholine	95.8	94.2	100	100
Adrenoceptors	92.8	98	100	98.2
Dopamine	98.2	100	100	96.4
Histamine	93.2	90	97.2	100
Trace amine	100	100	100	90
OACC	95.83	96.39	98.61	98.21

Table 4.8: ACC and OACC (both in %) of GPCR classification of class A subfamily Lipid at sub-subfamily level

Lipid	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
Cannabinoid	100	93.4	100	100
Free fatty acid	92	100	100	100
GPR18 - GPR55	68.4	61.4	76	100
Leukotriene	75	87	100	85.6
Lysophospholipid (LPA)	95	100	100	100
Lysophospholipid (S1P)	100	88.6	93.4	86.4
Platelet-activating factor	100	100	100	100
Prostanoid	97.5	100	100	100
OACC	92.56	93.02	97.67	96.74

Table 4.9: ACC and OACC (both in %) of GPCR classification of class A sub-family Nucleotide at sub-subfamily level

Nucleotide	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
Adenosine	100	100	100	100
P2Y	96.4	100	100	100
OACC	97.78	100	100	100

Table 4.10: ACC and OACC (both in %) of GPCR classification of class A sub-family Protein at sub-subfamily level

Protein	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
Chemerin	100	99	100	100
Chemokine	98	100	100	100
Glycoprotein homorne	100	100	100	100
Prokineticin	100	100	100	100
OACC	99.07	99.08	100	100

In all the sub-subfamily level classifications, Log-reg once again outperformed the rest. Even with its least performance among Lipid (under class A), it has an OACC of 97.67% as against 93.02% and 92.56% by SVM and PCA-ISA. In general, it produces an average OACC of approximately 99% as against 97% and 96% by SVM and PCA-ISA in this level.

4.3.5 Classification of GPCRs at Subtype level

The tables give the classification accuracies of the sub-subfamilies of class A GPCRs into its subtypes at fifth level together with the overall accuracies by the three classifiers.

Table 4.11: ACC and OACC (both in %) of GPCR classification of family A subfamily Alicaboxylic sub-subfamily 5-hydroxytryptamine at subtype level

5-hydroxytryptamine	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
5-HT$_{1A}$	100	100	100	100
5-HT$_{1B}$	100	100	96	76
5-HT$_{1D}$	100	80	100	100
5-HT$_{1E}$	50	60	60	76
5-HT$_{1F}$	100	90	66.8	88.4
5-HT$_{2A}$	78.4	95	96.6	76
5-HT$_{2B}$	81.8	40	60	40
5-HT$_{2C}$	100	100	100	100
5-HT$_{4}$	100	60	100	100
5-HT$_{5A}$	100	80	100	100
5-HT$_{7}$	100	65	100	100
OACC	94.54	91.82	96.51	96.36

Table 4.12: ACC and OACC (both in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Acetylcholine at subtype level

Acetylcholine	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
M$_1$	91.75	100	93.40	100
M$_2$	93.40	96.60	95	86.80
M$_3$	88.40	86.80	100	93.40
M$_4$	100	50	100	100
M$_5$	100	100	100	100
OACC	90.91	89.09	94.54	94.54

Table 4.13: ACC and OACC (both in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Adrenoceptors at subtype level

Adrenoceptors	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
α_{1A}	100	100	100	100
α_{1B}	100	93.40	100	100
α_{1D}	100	100	100	100
α_{2A}	80	100	70	90
α_{2B}	100	100	100	73.40
α_{2C}	100	60.80	100	100
β_1	83.40	91	100	85
β_2	100	100	100	85
β_3	100	100	100	100
OACC	96.19	91.43	97.14	93.33

Table 4.14: ACC and OACC (both in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Dopamine at subtype level

Dopamine	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
D_1	100	100	100	100
D_2	100	100	100	100
D_3	100	100	100	100
D_4	100	100	100	100
D_5	100	100	100	100
OACC	100	100	100	100

Table 4.15: ACC and OACC (both in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Histamine at subtype level

Histamine	**PCA-ISA**	**SVM**	**Log-Reg**	**MLP-NN**
H_1	100	90	100	83.40
H_2	100	100	100	100
H_3	95	85	100	100
H_4	94	75	100	100
OACC	97.14	85.71	100	94.29

Finally, in the subtype level, we again notice Log-Reg having the largest OACC of 96.51% followed closely by MLP-NN with 96.36% as against 94.1% and 91.82% by PCA-ISA and SVM.

4.4 Performance improvement by feature selection

Results from the previous sections indicated that Log-Reg and MLP-NN achieves better overall accuracies than the others using the full set of 1360 sequence features. We, therefore, use the f_classif function on these two proposed GPCR classifiers to see if we can further improve their performance by reducing the feature set. Our goal is to reduce the execution time while still maintaining similar or attaining even better accuracies.

4.4.1 Selected features

Table 4.16 indicates the average marginal contributions to the OACC across all the levels when some number of features were selected by the ANOVA FS technique described in Section 3.3.1 using results from the full set as reference.

Table 4.16: The contributions by features on Log-Reg and MLP-NN

Number of features selected	**Average marginal contributions (%)**
< 400	$\leq$ - 10
[400, 499]	$[-9, -6]$
[500, 599]	$[-5, -3]$
[600, 699]	$[-2, -1]$
[700, 799]	$[+0.8, +1]$
[800, 899]	$[+0.2, +0.7]$
[900, 999]	$[+0.05, +0.1]$
$\geq$ 1000	$[+0.0, +0.04]$

Now since the largest marginal increase was seen in the range with 700-799 features, I sought to identify which group of descriptors really had the majority of these features that contributed positively to accuracy improvement. It was found that those features belonged to DPC, QSO, PseAAC, Geary, and AAC (not in). This led to using each group of descriptors independently as main features to check the model performance (see Table 4.17 and Figure 4.3). Then extensive testing of various descriptor combinations were performed to see which combinations of these descriptors produced the best accuracies.

Table 4.17: The contributions by features on Log-Reg and MLP-NN

Descriptors	**Average performance (%)**	**Extraction time per sequence (secs)**
DPC	94.39	0.014
AAC	94.31	0.003
QSO	94.40	0.017
SOCN	81	0.013
GA	90.58	0.027
MA	90.97	0.022
NMBA	82	0.019
PseAAC	93.98	2.47

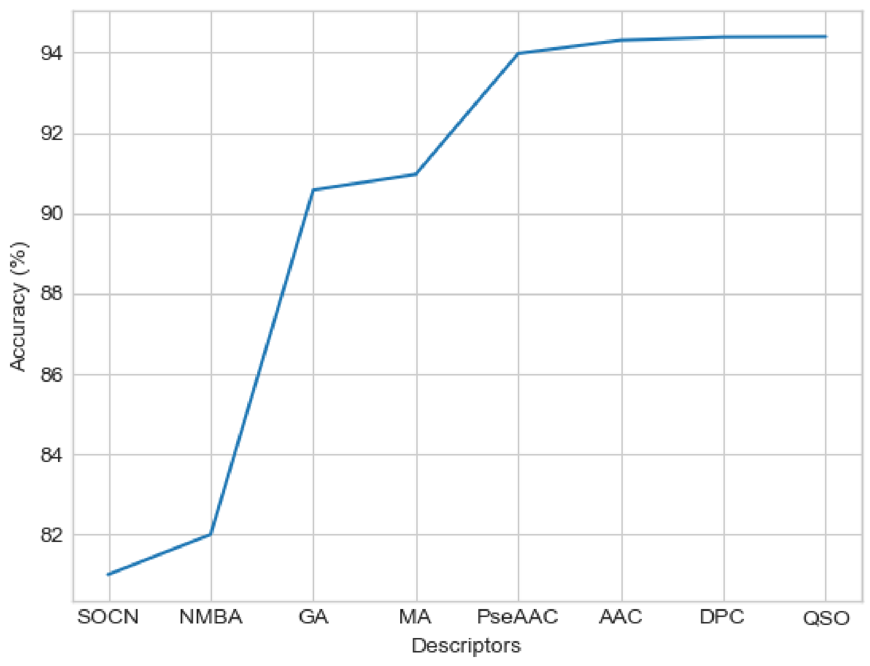


Figure 4.3: Descriptors performance.

4.4.2 Execution time

In order to optimize the classification performance, which means not only accuracies but also the efficiency or speed of execution, we look into the feature set. Table 4.18 also displays the time it takes to extract a feature set from a submitted sequence. Among those descriptors that had better performances, some had to be discarded based on the extraction time. After considering better performance both in terms of computational efficiency and accuracy, I ended up with four descriptors, DPC, QSO, GA, and AAC, which together form the reduced set of features. PseAAC provided a good performance but it had the worst time in the feature extraction stage so it was discarded. The final reduced group of descriptors has a vector with a length of 760 (=400 + 240 + 100 + 20) features, which falls within the desirable range of 700 - 799 features.

Now we compare the execution time for a given number of queries (unknown protein sequence(s)) to be predicted right from user submission through feature extraction to model classification stage between full set of 1360 features and the reduced set of 760 features under the descriptors DPC, QSO, GA, and AAC using the Log-Reg and MLP-NN. The times are the averages (μ) over five runs with its standard deviations (σ) in parentheses.

Table 4.18: Average and standard deviation of the execution times (secs) for Log-Reg

No. of proteins	**Reduced set time μ (σ)**	**Full set time μ (σ)**
1	25.17 (0.03)	42.93 (2.45)
50	26.88 (0.088)	212.25 (3.79)
100	34.89 (0.76)	586.34 (14.78)
200	46.99 (1.51)	1387.72 (10.91)
300	50.66 (1.09)	1989.62 (49.69)
400	55.83 (0.92)	2278.78 (93.41)
500	61.18 (3.46)	2745.14 (86.27)
1000	92.84 (1.31)	11559.3 (646.36)

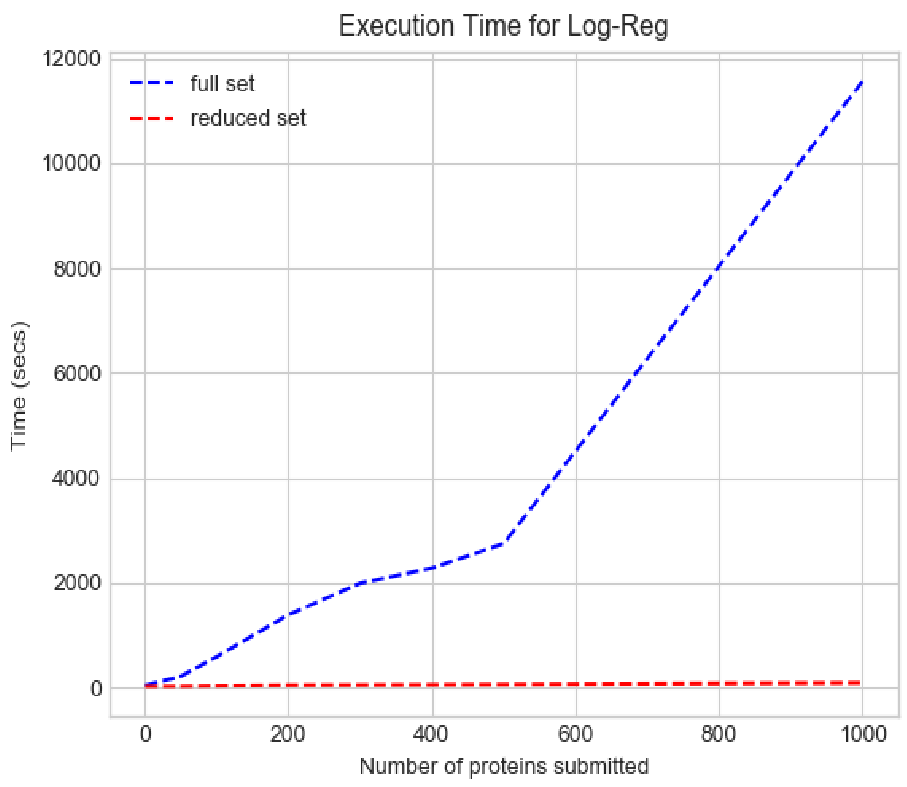


Figure 4.4: Execution time for Log-Reg.

Table 4.19: Average and standard deviation of the execution times (secs) for MLP-NN

No. of proteins	**Reduced set time** μ (σ)	**Full set time** μ (σ)
1	26.86 (0.47)	31.402 (0.20)
50	29.08 (0.61)	210.34 (3.04)
100	33.4 (0.96)	554.44 (8.87)
200	46.85 (0.96)	1378.11 (18.72)
300	51.03 (0.56)	1755.41 (18.71)
400	53.43 (0.61)	2229.68 (33.33)
500	63.59 (1.72)	2704.74 (11.60)
1000	95.08 (1.45)	11619.30 (15.50)

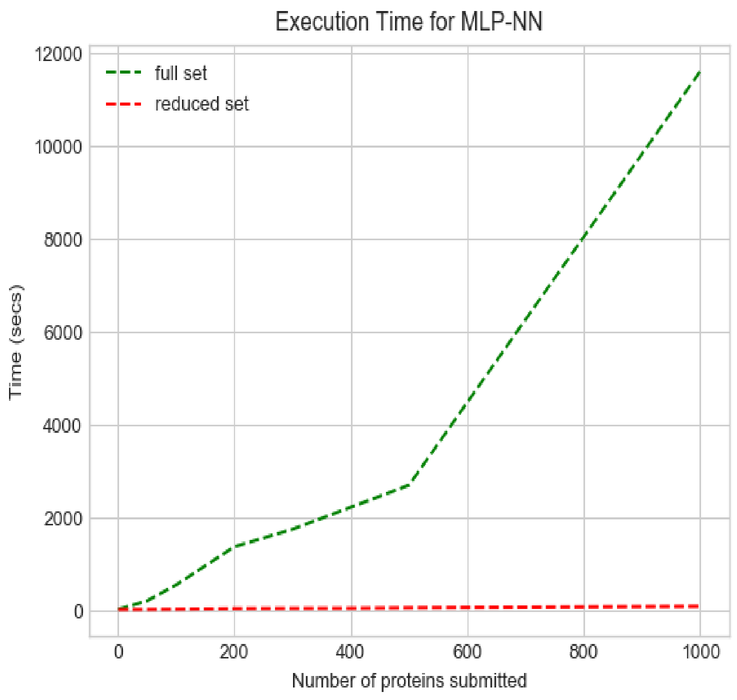


Figure 4.5: Execution time for MLP-NN

Let's consider only the execution time of reduced set of features using just Log-Reg classifier when over 10000 sequences are being submitted. It can be observed from Table 4.20 and Figure 4.6 that it takes less than 2 hrs for 80000 sequences to be classified by

Log-Reg and MLP-NN. If the full set of features had been used, the runtimes would be around two weeks.

Table 4.20: Average execution time (mins) of reduced set for Log-Reg & MLP-NN

No. of proteins	**Log-Reg time μ (σ)**	**MLP-NN time μ (σ)**
10000	10.96 (11.91)	12.14 (10.54)
20000	21.85 (11.83)	24.76 (11.90)
40000	44.97(14.93)	47.54 (16.67)
80000	88.87 (15.16)	94 (18.33)

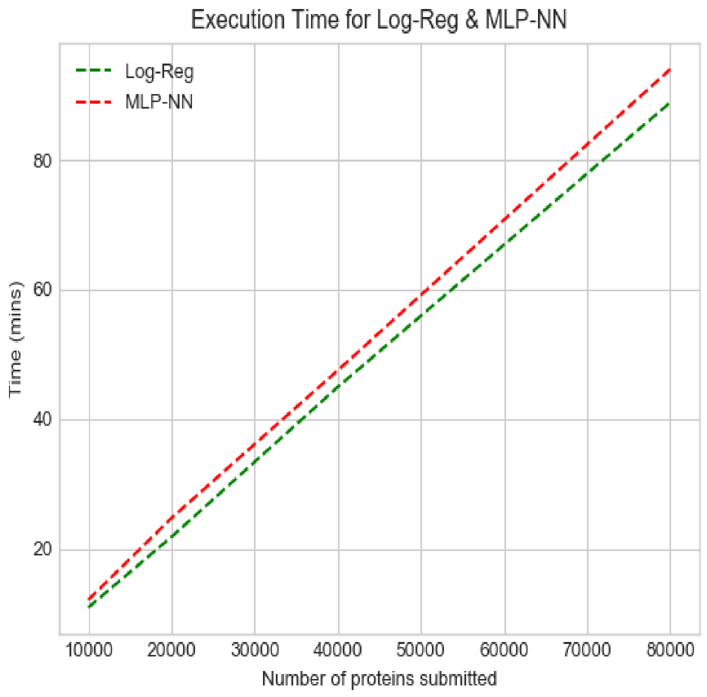


Figure 4.6: Execution time for Log-Reg & MLP

4.5 Results from SMOTE & Log-Reg

Table 4.22 represents results from combining synthetic minority oversampling technique (SMOTE) with penalized multinomial logistic regression (Log-Reg). The reported prediction accuracies from Table 4.4 for families D and E are generally lower than those of other families. Using this method then upsamples the minority classes to create a balanced dataset then we apply the Log-Reg as a classifier.

Table 4.21: ACC and OACC classification of GPCR at family level

Family	**SMOTE + Log-Reg**
Class A	100
Class B1	100
Class B2	100
Class C	98.9
Class D	97.5
Class E	99
Class F	99
Class T2R	100
OACC	**99.6**

Also, it has been noticed that when the original features were reduced by leaving out redundant and irrelevant ones, there was an improvement in the overall performance and in the over all execution time (as seen in figures 4.4 and 4.5) . That is, when informative features were been used, computational efficiency improves significantly. This gives us insight of how the existence of noise affects the general performance. Again, we notice that using SMOTE to handle minority classes helps to generate a balanced set for the best performance to be achieved.

4.6 Discussion

Classification results from other families to their various subtypes performed similarly with MLP-NN and Log-Reg outperforming the rest of the algorithms as shown in the appendix.

In demonstrating the superiority in performance, we made comparisons with a number of existing online web-servers which were trained from datasets different from what I used. A balanced independent dataset was used to test the performance of these web-servers and the results showed that;

1. For GPCRpred, the overall accuracy at level 1, was found to be less than 90% with FPR of 36.83% among the transmembrane non-GPCRs. Levels 2 and 3 were between 80% - 90%.

2. For PCA-GPCR, FPR among transmembrane non-GPCR was 70+% with the lowest overall accuracy of 83% even though it had the best TPR (99%) compared to the other servers. All other levels were between 85% and 92%.

3. And for GPCR-CA, the overall accuracy at level 1 was 88.68% with a FPR of 27.92% among transmembrane non-GPCRs. But at level 2, had an accuracy of 91.30%.

Chapter 5

Web-server

There are existing GPCR protein sequence prediction web-servers for most of the machine learning algorithms except few ones like Log-Reg and MLP-NN. Even with those available ones some of which are, PCA-GPCR, GPCRPred, SVMProt, GPCR-CA etc still have some limitations. The user is not required to have any knowledge of machine learning to be able to use this. This chapter highlights the web-server developed from the two proposed algorithms with the aim of tackling the limitations associated with the existing tools.

5.1 GPCR-PEn web-server

GPCR-PEn, which represents GPCR Prediction Ensemble, is an easy to use pipeline developed by our team with software tools which includes a database and prediction tools to facilitate the prediction and classification of GPCRs.

The web-server based on python, apache and webpy. Apache (cross-platform web-server software) interacts with the web.py using Web Server Gateway Interface (WSGI). The webpy gpcrSubmitServer script then creates a parameter file and then calls a gpcr command line script to process the sequences. The site then checks the log file to see if the analysis is completed. Once it is completed, the webpy then reads the csv of the command line script to generate the table on the results page.

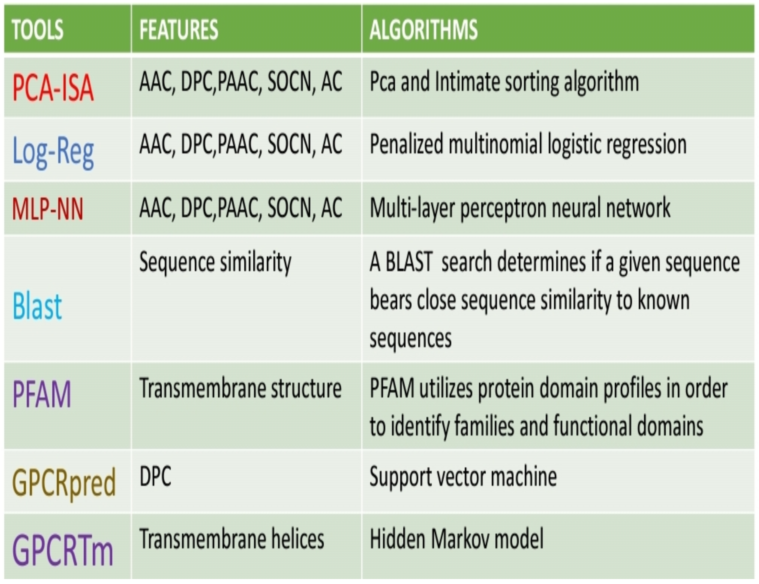
The associated database tool, GPCR-PEnDB, is a relational database which has data for GPCR and non-GPCR protein sequences including their source organisms, their accession IDs, the various levels they belong to and their functions. These data can be collected and used as a useful training and testing dataset for various analysis.

5.2 Prediction tools

There are four existing tools that have been locally installed and implemented in the pipeline by our research team and they are Blast, PFAM, GPCRpred and GPCRTm. These tools have various pitfalls and that resulted in proposing the logistic regression (Log-Reg) and multi-layer perceptron neural network (MLP-NN) which. I have incorporated, PCA-ISA was also included since its existing web-server (PCA-GPCR) is no longer functional.

In total, the pipeline now has seven prediction tools with different algorithm each. But in terms of performance, the three new additions out-performs the rest. They also classify the GPCRs to its lowest level which is the subtype whereas the others do not.

The algorithms utilize sequence similarity, transmembrane structure, and sequence derived features such as amino acid composition, dipeptide composition, autocorrelations and order coupling number to determine if a protein sequence is a GPCR. Below is a table of the tools and their respective features and algorithms;



TOOLS	FEATURES	ALGORITHMS
PCA-ISA	AAC, DPC,PAAC, SOCN, AC	Pca and Intimate sorting algorithm
Log-Reg	AAC, DPC,PAAC, SOCN, AC	Penalized multinomial logistic regression
MLP-NN	AAC, DPC,PAAC, SOCN, AC	Multi-layer perceptron neural network
Blast	Sequence similarity	A BLAST search determines if a given sequence bears close sequence similarity to known sequences
PFAM	Transmembrane structure	PFAM utilizes protein domain profiles in order to identify families and functional domains
GPCRpred	DPC	Support vector machine
GPCRTm	Transmembrane helices	Hidden Markov model

Figure 5.1: The Pipeline Prediction tools .

5.2.1 How the proposed tools work

Anytime a query is being submitted, the following is a flow of how the tools work.
The text data is first converted to fixed-length of features which are the reduced set of features known as DPC, QSO, GA, and AAC then the saved model from the algorithm on a training set with same features is applied for prediction. This process happens in the background, the only thing the user sees is the output of results. A schematic diagram of the workflow is shown below;

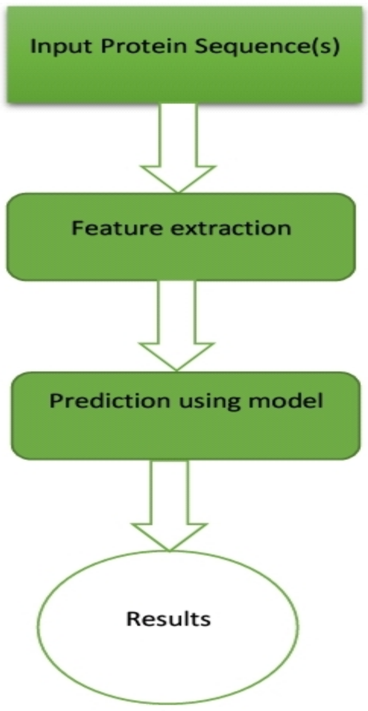
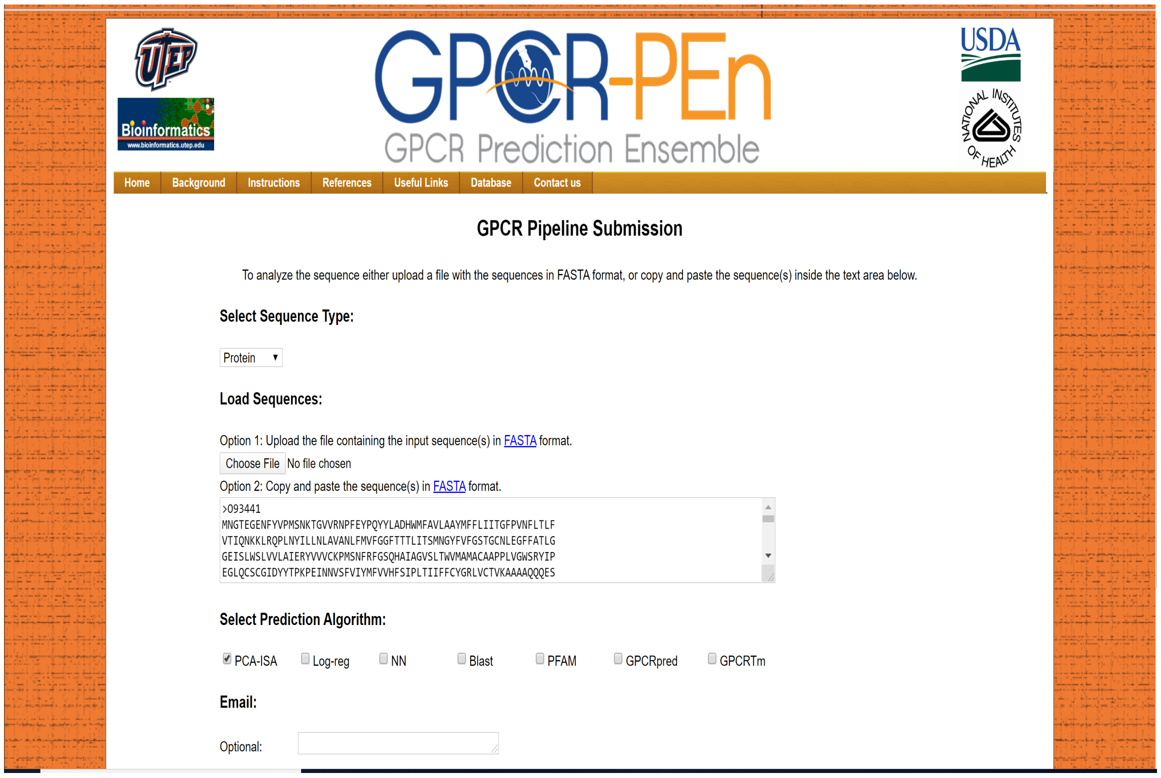


Figure 5.2: Protein Sequence Workflow

5.2.2 How to use it

The user has the option to either upload a fasta file or or pasting a fasta sequence into the text with any number of sequences to be analyzed. Once they have been uploaded, any of the tools can then be selected but to get an optimized results, it's recommended to select all tools then majority votes can be implemented on the results produced. Then the "process sequence" button is clicked to process the uploaded sequences which could take couple of minutes depending on the number of sequences. Below, in Figure 5.3, is the user-friendly graphical user interface of the prediction tool page from GPCR-PEn pipeline;



GPCR Pipeline Submission

To analyze the sequence either upload a file with the sequences in FASTA format, or copy and paste the sequence(s) inside the text area below.

Select Sequence Type:

Protein

Load Sequences:

Option 1: Upload the file containing the input sequence(s) in FASTA format.

Choose File No file chosen

Option 2: Copy and paste the sequence(s) in FASTA format.

```
>093441
MNGTEGENFYVPMSNKTGVVRNPFEYPQYYLADHWMFAVLAAYMFFLIITGFPVNFLTLF
VTIQNKKLRQPLNYILLNLAVANLFMVFGGFTTTLITSMNGYFVFGSTGCNLEGFFATLG
GEISLWSLVVLAIERYVVVCKPMSNFRFGSQHAIAGVSLTWVMAMACAAPPLVGWSRYIP
EGLQCSCGIDYYTPKPEINNVSFVIYMFVVHFSIPLTIIFFCYGRLVCTVKAAAAQQQES
```

Select Prediction Algorithm:

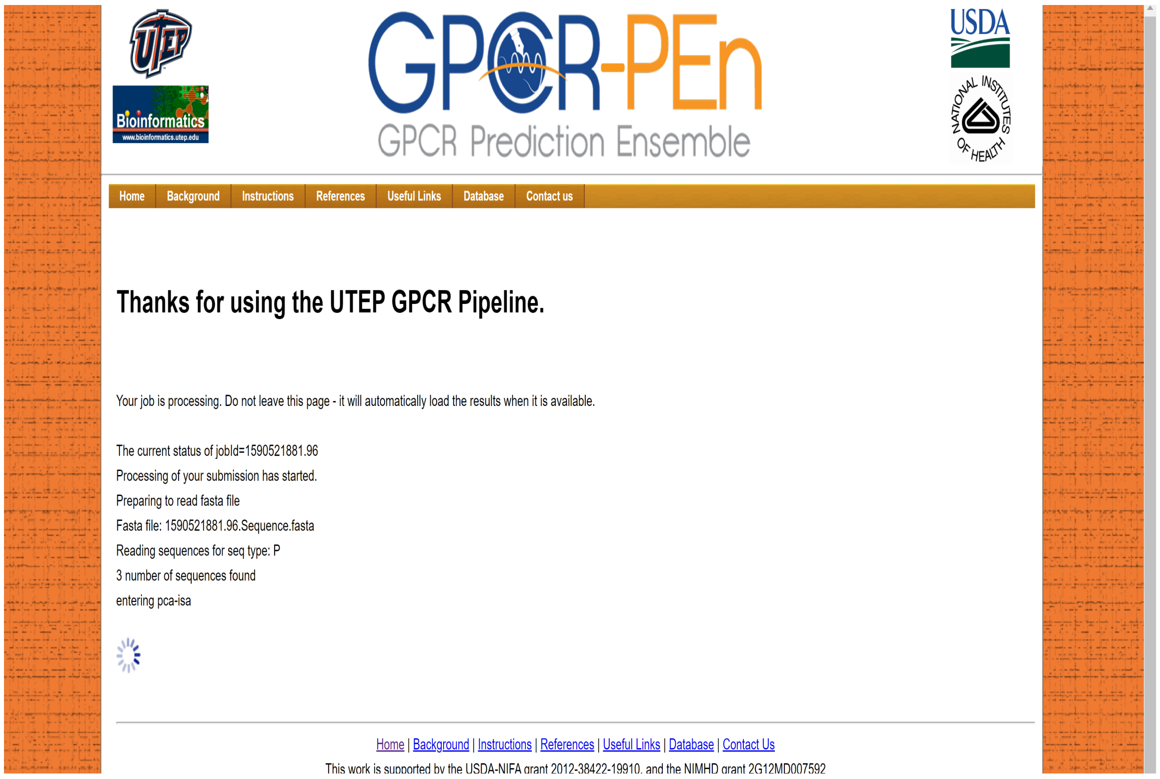
[✓ PCA-ISA] Log-reg NN Blast PFAM GPCRpred GPCRTm

Email:

Optional:

Figure 5.3: The submission page.

An image of the processing page is found below;



Thanks for using the UTEP GPCR Pipeline.

Your job is processing. Do not leave this page - it will automatically load the results when it is available.

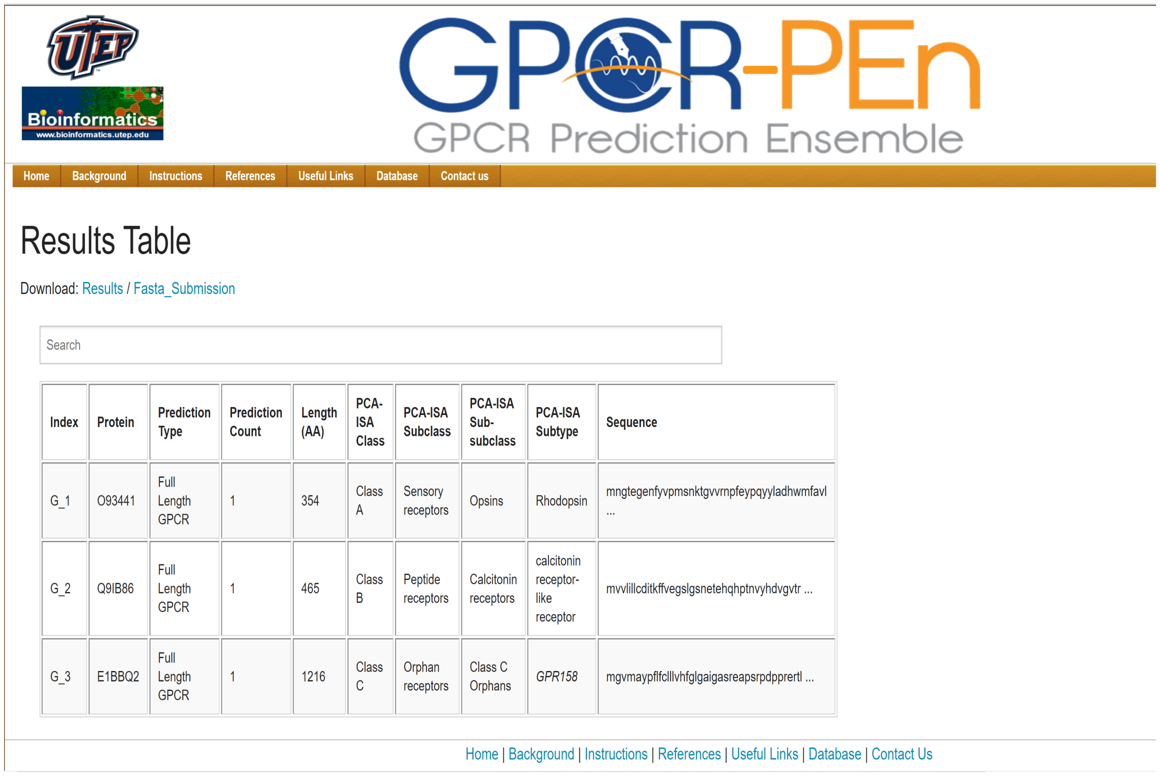
```
The current status of jobId=1590521881.96
Processing of your submission has started.
Preparing to read fasta file
Fasta file: 1590521881.96.Sequence.fasta
Reading sequences for seq type: P
3 number of sequences found
entering pca-isa
```

Home | Background | Instructions | References | Useful Links | Database | Contact Us

This work is supported by the USDA-NIFA grant 2012-38422-19910, and the NIMHD grant 2G12MD007592

Figure 5.4: The processing page.

After which a display page shows the output in a table as shown below;



Results Table

Download: Results / Fasta_Submission

Search

Index	Protein	Prediction Type	Prediction Count	Length (AA)	PCA-ISA Class	PCA-ISA Subclass	PCA-ISA Sub-subclass	PCA-ISA Subtype	Sequence
G_1	O93441	Full Length GPCR	1	354	Class A	Sensory receptors	Opsins	Rhodopsin	mngtegenfyvpmsnktgvvrnpfeypqyyladhwmfavl ...
G_2	Q9IB86	Full Length GPCR	1	465	Class B	Peptide receptors	Calcitonin receptors	calcitonin receptor-like receptor	mvvlillcditkffvegslgsnetehqhptnvyhdvgvtr ...
G_3	E1BBQ2	Full Length GPCR	1	1216	Class C	Orphan receptors	Class C Orphans	*GPR158*	mgvmaypflfclllvhfglgaigasreapsrpdpprertl ...

Home | Background | Instructions | References | Useful Links | Database | Contact Us

Figure 5.5: The results page.

The columns of the output page are;

Index: Number of sequences submitted

Protein: Accession ID

Protein Type: Full length or partial length (fragment)

Prediction Count: Number of tools out of total selected that classified as GPCR

Models selected for sequence classification

Sequence: Protein Sequence

This web-based tool is currently under beta-testing on the website accessible within the UTEP (VPN) virtual private network at http://biolinux19.bioinformatics.utep.edu/fayivor. It will be part of the major upgrade of our GPCR-PEn web-server that will be made public by August 2020.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The experimental results show that MLP-NN and SVM have, on average, the lowest false positive rate with the best overall performances at the superfamily level then followed closely by Log-Reg. At the family level, MLP-NN and Log-Reg outperforms the rest with the same overall accuracy. But from the sub-family, sub-subfamily to subtype levels, Log-Reg produces the best performance which is closely followed by MLP-NN then SVM. This shows that, both MLP-NN and Log-Reg are two important classifiers that can most accurately classifying GPCRs to the subtype level. Furthermore, during the prediction and testing processes, we were able to determine the contribution and importance of the descriptors in accurately classifying GPCRs. The top four descriptors are QSO, DPC, AAC, and PseAAC in decreasing order of importance. As it was necessary to implement several new codes in order to perform the analyses on our local computer network, several improvements over existing GPCR prediction and classification tools have been observed. First, the new PCA-ISA program has a major advantage over the existing PCA-GPCR tool that uses a similar algorithm. PCA-ISA has very high capabilities of distinguishing GPCRs from transmembrane non-GPCRs with a much lower FPR of 10% compared to 70+% by PCA-GPCR. Second, the results from SMOTE further buttress the success in achieving better classification accuracy from large balanced training data. Finally, with the incorporation of the rank-based feature selection method, a non-redundant feature set has been obtained for developing a web-server that can take over 10,000 sequence queries at one time and efficiently produce results within seconds of submission, a substantial improvement over existing tools.

6.2 Future research goals and proposed approaches

While this dissertation project resulted in a web-based tool that can accurately and efficiently classify GPCRs from superfamily level to subtype level using only protein sequence information, it can be extended in several directions:

1. Incorporate 3D structural features to the sequence-derived features to further reduce the false positive rates and help produce a good model for classifying other types of transmembrane proteins.

2. Survey and assess other deep learning algorithms, such as convolutional neural networks, which may produce better performance than that of MLP-NN. Delving more into these deep learning algorithms [Li M. et al., 2018] can be of immense importance in improving the accuracy to almost 100%.

3. Further analysis on the feature selection step can be done by considering other rank-based methods and just using subsets of features within the descriptors. For example, instead of using all 20 features from the amino acid composition descriptor, we can just select a few most significant ones. The same applies to all other descriptors. This helps in reducing the computational time for the classification process so that very big datasets can be handled at one time.

4. Identification of ligand binding residues is important for understanding the biological functions of GPCRs. As GPCRs are popular therapeutic drug targets, these computational methods might be extended and used as sequence-based methods for predicting GPCR-ligand binding residues. Using these same combination of descriptors for GPCRs, some attributes from protein ligands can be extracted then the classifiers applied.

I anticipate that these future investigations will result in highly effective computational tools that can be conveniently disseminated to the GPCR research community worldwide via our web server at gprc.utep.edu.

References

[1] Altschul S., Gish W., Miller W., and Lipman D., (1990) Basic local alignment search tool. J. Mol. Biology 215, 403-410.

[2] Anagnostopoulos, C., Tasoulis, D.K., Adams, N.M., Pavlidis, N.G., Hand, D.J.: Online linear and quadratic discriminant analysis with adaptive forgetting for streaming classification. Stat. Anal. Data Mining 5(2), 139–166 (2012).

[3] Attwood,T.K. (2001) A compendium of specific motifs for diagnosing GPCR subtypes. Pharmacol Sci., 22,162-5.

[4] Baud V, Chissoe SL, Viegas-Pequignot E, Diriong S, N'Guyen VC, Roe BA, Lipinski M. EMR1, an unusual member in the family of hormone receptors with seven transmembrane segments. Genomics. 1995;26:334–344. The cloning of a prototype member of subfamily B2.

[5] Begum, K., Mohl, J. E., Ayivor F., Perez E., and Leung, M.-Y. (2020), The Border Biomedical Research Center, "GPCR-PEnDB: A database of protein sequences and derived features to facilitate prediction and classification of G protein-coupled receptors," Database Journal of Oxford.

[6] Begum, K., Mohl, J. E., and Leung, M.-Y., Sixteenth Annual Rocky Mountain Bioinformatics Conference 2018, Aspen/Snowmass, Colorado, "GPCR-PEnDB: A database of protein sequences and derived features to facilitate prediction and classification of G protein-coupled receptors," International Society for Computational Biology. (December 6, 2018).

[7] Bettler, B.; Kaupmann, K.; Mosbacher, J. and Gassmann, M. (2004) Physiol. Rev., 84, 835-867.

[8] Bhasin, M. and Raghava, G.P. 2004 Classification of nuclear receptors based on amino acid composition and dipeptide composition J. Biol. Chem . 279 23262 –23266.

[9] 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.

[10] Bock, J.R. and Gough, D.A. 2001 Predicting protein–protein interactions from primary structure Bioinformatics 17 455 –460.

[11] Bräuner-Osborne, H., Wellendorf, P. & Jensen, A. A. Structure, pharmacology and therapeutic prospects of family C G-protein coupled receptors. Curr. Drug Targets 8, 169–184 (2007).

[12] Breiman L. Bagging predictors. Machine Learning, 24 (2):123–140, 1996. 18

[13] Broto P, Moreau G, Vandicke C: Molecular structures: perception, autocorrelation descriptor and SAR studies. Eur J Med Chem 1984, 19: 71–78.

[14] Bryson-Richardson, Darren Logan, Peter David Currie, Ian J Jackson. (2004). Large-scale analysis of gene structure in rhodopsin-like GPCRs: Evidence for widespread loss of an ancient intron.

[15] Cai CZ, Han LY, Ji ZL, Chen X, Chen YZ: SVM-Prot: web-based support vector machine software for functional classification of a protein from its primary sequence. Nucleic Acids Res. 2003, 31 (13): 3692-3697. 10.1093/nar/gkg600.

[16] Cai, Y., Zhou, G. (2003). Support Vector Machines for Predicting Membrane Protein Types by Using Functional Domain Composition, 84(5): 3257–3263. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1302886/

[17] Chen, Y.-W. , & Lin, C.-J. (2003). Combining SVMs with various feature selection strategies. In NIPS 2003 feature selection challenge (pp. 1–10) .

[18] 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.

[19] Chou, K.C. 2000 Prediction of protein subcellular locations by incorporating quasi-sequence-order effect Biochem. Biophys. Res. Commun . 278 477 –483

[20] 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.

[21] Chou KC (2005a). Prediction of G-protein-coupled receptor classes. J. Proteome Res., 2005, 4 (4), pp 1413–1418.

[22] 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.

[23] Cohen G., H. Sax, A. Geissbuhler, Novelty detection using one-class parzen density estimator: an application to surveillance of nosocomial infections stud. Health Technol Inform 136 (2008) 21-26.

[24] Cooper GM. Sunderland (MA) (2000). The Cell: A Molecular Approach. 2nd edition. IEEE/ACM https://doi.org/10.1109/TCBB.2010.101.

[25] Couvineau A, Laburthe M (2012a). VPAC receptors: structure, molecular pharmacology and interaction with accessory proteins. Br J Pharmacol 166: 42–50.

[26] 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.

[27] Dijck V., P. Nutrient sensing G protein-coupled receptors: interesting targets for antifungals? Med. Mycol. 47, 671–680 (2009).

[28] Dingledine, R.; Borges, K.; Bowie, D. and Traynelis, S.F. (1999) Pharmacol. Rev., 51, 7-61.

[29] Dong-Sheng Cao, Qing-Song Xu, Yi-Zeng Liang, propy: a tool to generate various modes of Chou's PseAAC, Bioinformatics, Volume 29, Issue 7, 1 April 2013, Pages 960–962, https://doi.org/10.1093/bioinformatics/btt072.

[30] Duda R., P Hart, and D. Stork Pattern Classification. Wiley 2000.

[31] Durbin R., Eddy S., Krogh A., and Mitchison G., (1998) Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids Cambridge University Press, New York.

[32] 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), 8959–75. https://doi.org/10.1021/bi047316u

[33] 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.

[34] Feng, Z.P. and Zhang, C.T. 2000 Prediction of membrane protein types based on the hydrophobic index of amino acids J. Protein Chem . 19 269 –275.

[35] Frank,E. and Witten,I.H. (1998) Generating Accurate Rule Sets without Global Optimization. Fifteenth International Conference on Machine Learning.

[36] Fredriksson, R. et al (2003). The G-protein-coupled receptors in the human genome form five main families. Phylogenetic analysis, paralogon groups, and fingerprints. Molecular Pharmacology;63(6):1256-72. https://www.ncbi.nlm.nih.gov/pubmed/12761335

[37] Fridmanis, D. et al. (2006) Formation of new genes explains lower intron density in mammalian Rhodopsin G protein-coupled receptors. Mol Phylogenet Evol., 43, 864-80.

[38] 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

[39] Geary RC: The contiguity ratio and statistical mapping. Incorp Statist 1954, 5: 115–145. 10.2307/2986645

[40] Gether,U. et al. (2002) Structural basis for activation of G-proteincoupled receptors. Pharmacol Toxicol., 91, 304-312.

[41] Gloriam, D.E. et al. (2005) Nine new human Rhodopsin family G protein coupled receptors: identification, sequence characterisation and evolutionary relationship. Biochim Biophys Acta, 1722, 235-46

[42] Goudet C, et al. Heptahelical domain of metabotropic glutamate receptor 5 behaves like rhodopsin-like receptors. Proc Natl Acad Sci U S A. 2004;101:378–383

Guo-Sheng Han, Zu-Guo Yu, Vo Anh

[43] Grassmann, J., Reczko, M., Suhai, S., Edler, L. 1999 Protein fold class prediction: new methods of statistical classification Proc. Int. Conf. Intell. Syst. Mol. Biol . 106 –112

[44] 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

[45] 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 pro-

tein power spectrum from fast Fourier transform. Amino Acids, 30(4), 397–402. https://doi.org/10.1007/s00726-006-0332-z

[46] Guo,Y.Z. et al. (2005) Fast fourier transform-based support vector machine for prediction of G-protein coupled receptor subfamilies Acta Biochim Biophys Sin (Shanghai), 37, 759-66.

[47] Grantham, R. 1974 Amino acid difference formula to help explain protein evolution Science 185 862 –864

[48] 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.

[49] Hamm H. The many faces of G-protein signaling. J Biol Chem. 1966;273:669–672.

[50] Han, L.Y., Cai, C.Z., Ji, Z.L., Cao, Z.W., Cui, J., Chen, Y.Z. 2004 Predicting functional family of novel enzymes irrespective of sequence similarity: a statistical learning approach Nucleic Acids Res . 32 6437 –6444

[51] Harmar, A. J. Family-B G-protein-coupled receptors. Genome Biol. 2, Reviews3013.1–3113.10 (2001)

[52] Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer.

[53] Hauser A.S., Attwood M.M., Rask-Andersen M., Schiöth H.B., Gloriam D.E. Trends in GPCR drug discovery: new agents, targets and indications. Nat. Rev. Drug Discov. 2017; 16:829–842.

[54] Hebert,T.E. and Bouvier,M. (1998) Structural and functional aspects of G protein-coupled receptor oligomerization. Biochem Cell Biol. 76, 1-11

[55] 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

[56] 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

[57] Horne, D.S. 1988 Prediction of protein helix content from an autocorrelation analysis of sequence hydrophobicities Biopolymers 27 451 –477

[58] Hua, S. and Sun, Z. 2001 Support vector machine approach for protein subcellular localization prediction Bioinformatics 17 721 –728

[59] 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

[60] 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

[61] 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

[62] 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

[63] I. T. Jolliffe, Principal Component Analysis (Springer-Verlag, New York, 1989)

[64] James G, Witten D,Hastie T, Tibshirani R. Treebased methods. An Introduction to Statistical Learning with Applications in R. 1st ed. New York, Springer Science + Business Media, 2013, p. 303

[65] Jensen, A.A. (2004) in Molecular Neuropharmacology. Strategies and Methods, (A. Schousboe and H. Bräuner-Osborne), Humana Press, pp. 47-82.

[66] Karchin, R., Karplus, K., Haussler, D. 2002 Classifying G-protein coupled receptors with support vector machines Bioinformatics 18 147 –159

[67] Kaupmann, K.; Huggel, K.; Heid, J.; Flor, P.J.; Bischoff, S.; Mickel, S.J.; McMaster, G.; Angst, C.; Bittiger, H.; Fröstl, W. and Bettler, B. (1997) Nature, 386, 239-246

[68] Kawashima S, Kanehisa M. AAindex: amino acid index database, Nucleic Acids Res., 2000, vol. 28 pg. 374

[69] Keerthi, S.S. et al. (2001) Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation, 13, 637-649.

[70] Klabunde,T. and Hessler,G. (2002) Drug design strategies for targeting G-protein-coupled receptors. ChemBioChem 2002, 3, 928– 944.

[71] Kingsford, C. & Salzberg, S.L. What are decision trees? Nat. Biotechnol. 26, 1011–1013 (2008).

[72] Kochman, K. Superfamily of G-protein coupled receptors (GPCRs) – extraordinary and outstanding success of evolution. Postepy Higieny I Medycyny Doswiadczalnej 68, 1225–1237 (2014)

[73] Laburthe M, Couvineau A, Tan V. Class II G protein-coupled receptors for VIP and PACAP: structure, models of activation and pharmacology. Peptides 2007; 28: 1631-39

[74] 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

[75] Li L., Cui X., Yu S., Zhang Y., Luo Z., Yang H., Zhou Y., Zheng X. PSSP-RFE: accurate prediction of protein structural class by recursive feature extraction from

PSI-BLAST profile, physical-chemical property and functional annotations PLoS One, 9 (2014), Article e92863.

[76] Liao, Z., Ju, Y., and Zou, Q. (2016). Prediction of G protein-coupled receptors with SVM-Prot features and random forest. Scientifica 2016:8309253. doi: 10.1155/2016/8309253

[77] Lin Y.Q., Min X.P., Li L.L., Yu H., Ge S.X., Zhang J., Xia N.S. Using a machine-learning approach to predict discontinuous antibody-specific B-cell epitopes Curr. Bioinform., 12 (2017), pp. 406-415

[78] Li, M., Ling, C., Xu, Q. et al. Classification of G-protein coupled receptors based on a rich generation of convolutional neural network, N-gram transformation and multiple sequence alignments. Amino Acids 50, 255–266 (2018). https://doi.org/10.1007/s00726-017-2512-4

[79] 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

[80] 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

[81] 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

[82] 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

[83] Lin, Z. and Pan, X.M. 2001 Accurate prediction of protein secondary structural content J. Protein Chem . 20 217 –220

[84] Li ZR, et al. PROFEAT: a web server for computing structural and physicochemical features of proteins and peptides from amino acid sequence, Nucleic Acids Res. , 2006, vol. 34 (pg. W32-W37)

[85] Lin WZ, Xiao X, Chou KC. GPCR-GIA: a web-server for identifying G-protein coupled receptors and their families with grey incidence analysis. Protein Eng Des Sel. 2009;22:699–705.

[86] Mafteiu-Scai L.O. A new dissimilarity measure between feature-vectors International Journal of Computer Applications, 64 (17) (2013), pp. 39-44.

[87] Malbon CC (2004). "Frizzleds: new members of the superfamily of G-protein-coupled receptors". Front. Biosci. 9: 1048–58. doi:10.2741/1308. PMID 14977528

[88] McKnight AJ, Gordon S. EGF-TM7: a novel subfamily of seven-transmembrane-region leukocyte cell-surface molecules. Immunol Today. 1996;17:283–287. A review of the structural diversity and possible immunological functions of receptors in subfamily B2, here referred to as 'EGF-TM7' receptors.

[89] Moran, P.A. 1950 Notes on continuous stochastic phenomena Biometrika 37 17 –23

[90] Munk, C. et al. GPCRdb: the G protein-coupled receptor database - an introduction. Br. J. Pharmacol. 173, 2195–2207 (2016)

[91] Nakayama N, Miyajima A, Arai K. EMBO J., Nucleotide sequences of STE2 and STE3, cell type-specific sterile genes from Saccharomyces cerevisiae. 2643-2648, (1985).

[92] 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

[93] N.V. Chawla, K.W. Bowyer, L.O. Hall, & W.P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique" J. Artificial Intelligence Research, vol. 16, pp. 321-357, 2002

[94] Ong, S.A., Lin, H.H., Chen, Y.Z. et al. Efficacy of different protein descriptors in predicting protein functional families. BMC Bioinformatics 8, 300 (2007). https://doi.org/10.1186/1471-2105-8-300

[95] 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

[96] Papasaikas PK, Bagos PG, Litou ZI, Promponas VJ, Hamodrakas SJ. PRED-GPCR: GPCR recognition and family classification server. Nucleic Acids Res 2004; 32:380 - 382

[97] 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), 2444–8. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/3162770

[98] Pedregosa,F., & Varoquaux,G., & Gramfort,A., & Michel,V., Thirion,B., Grisel,O., Blondel,M., Prettenhofer,P., Weiss,R., Dubourg,V. et al. (2011) Scikit-learn: machine learning in python. J. Mach. Learn. Res., 12, 2825–2830

[99] 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

[100] Perez, E., Begum K, Mohl, J., & Leung, M., (2018). Assessment of web-based G protein-coupled receptor prediction and classification bioinformatics tools. 23rd Joint UTEP/NMSU Conference on Mathematics, Computer Science and Computational Sciences. http://www.cs.utep.edu/vladik/utepnmsu18.html

[101] Poluliakh, N., Saito, K., Shimizu, T., 2000. Transmembrane topology pattern and detection of transmembrane protein functions. In: Dunker, A.K., Konagaya, A., Miyano, S., Takagi, T. (Eds.), Genome Informatics 2000. Universal Academy Press, Tokyo, pp. 334–335.

[102] 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

[103] 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

[104] QingJun Song, HaiYan Jiang & Jing Liu (2017). Feature selection based on FDA and F-score for multi-class classification.Expert Systems with Applications, 81 (2017), pp. 22-27

[105] 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), 433–45. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/14732403

[106] Reczko, M. and Bohr, H. 1994 The DEF data base of sequence based protein fold class predictions Nucleic Acids Res . 22 3616 –3619

[107] Rios, S., Fernandez, M.F., Caltabiano, G. et al. GPCRtm: An amino acid substitution

matrix for the transmembrane region of class A G Protein-Coupled Receptors. BMC Bioinformatics 16, 206 (2015) doi:10.1186/s12859-015-0639-4

[108] Ruiz-Gómez A, Molnar C, Holguín H, Mayor F, de Celis JF (April 2007). "The cell biology of Smo signalling and its relationships with GPCRs". Biochimica et Biophysica Acta. 1768 (4): 901–12. doi:10.1016/j.bbamem.2006.09.020. PMID 17094938

[109] 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

[110] Schneider, G. and Wrede, P. 1994 The rational design of amino acid sequences by artificial neural networks and simulated molecular evolution: de novo design of an idealized leader peptidase cleavage site Biophys. J . 66 335 –344

[111] Shengli Zhang & Xin Duan (2017). Prediction of protein subcellular localization with oversampling approach and Chou's general PseAAC.https://doi.org/10.1016/j.jtbi.2017.10.030

[112] Shepherd, A.J., Gorse, D., Thornton, J.M. 2003 A novel approach to the recognition of protein architecture from sequence using Fourier analysis and neural networks Proteins 50 290 –302

[113] Singh G, Inoue A, Gutkind JS, Russell RB, Raimondi F. PRECOG: PREdicting COupling probabilities of G-protein coupled receptors Nucl Acids Res. (Accepted), 2019.

[114] Spiegel,A.M., Shenker,A. and Weinstein,L.S. (1992) Endocr. Rev., 13, 536–565

[115] Sokal, R.R. and Thomson, B.A. 2006 Population structure inferred by local spatial autocorrelation: an example from an Amerindian tribal population Am. J. Phys. Anthropol . 129 121 –131

[116] Song, QingJun & Jiang, HaiYan & Liu, Jing. (2017). Feature selection based on FDA and F-score for multi-class classification. Expert Systems with Applications. 81. 10.1016/j.eswa.2017.02.049.

[117] Souza, L. Rittner, Lotufo A comparison between k-Optimum Path Forest and k-Nearest Neighbors supervised classifiers Pattern Recognition Letters, 39 (2014), pp. 2-10.

[118] Srivastava S., M.R. Gupta, B.A. Frigyik Bayesian quadratic discriminant analysis Journal of Machine Learning Research, 8 (June) (2007), pp. 1277-1305

[119] Stacey M, Lin HH, Gordon S, McKnight AJ. LNB-TM7, a group of seven-transmembrane proteins related to family-B G-protein-coupled receptors. Trends Biochem Sci. 2000;25:284–289. A review of the structures and functions of receptors in subfamily B2, here referred to as 'LNB-TM7' receptors.

[120] Strader, C.D., Fong,T.M., Tota,M.R. and Underwood,D. (1994) Annu. Rev. Biochem., 63, 101–132.

[121] Sugiyama Y., N. Poluliakh, T. Shimizu Identification of transmembrane protein functions by binary topology patterns Protein Eng., 16 (2003), pp. 479-488

[122] Tong, S. & Koller, D. Support vector machine active learning with applications to text classification. J. Mach. Learn. Res. 2 (Ma. 2002), 45-66

[123] Tuteja N. Signaling through G protein coupled receptors. Plant Signal Behav. 2009;4(10):942–947. doi:10.4161/psb.4.10.9530

[124] Ulloa-Aguirre A, Zariñán T, Dias JA, Conn PM. Mutations in G protein-coupled receptors that impact receptor trafficking and reproductive function. Mol Cell Endocrinol. 2014;382(1):411–423. doi:10.1016/j.mce.2013.06.024

[125] Xiao X, Wang P, Chou KC. GPCR-CA: A cellular automaton image approach for predicting G-protein-coupled receptor functional classes. Journal of Computational Chemistry. 2009;30:1414–1423. https://www.ncbi.nlm.nih.gov/pubmed/19037861

[126] V. Vapnik, “The nature of statistical learning theory,” Springer-Verlag: New York, 1995

[127] Van der Knaap J.A, C.P. Verrijzer Undercover: gene control by metabolites and metabolic enzymes Genes Dev., 30 (2016), pp. 2345-2369

[128] Venkatakrishnan AJ, Ma AK, Fonseca R, Latorraca NR, Kelly B, Betz RM, Asawa C, Kobilka BK, Dror RO (2019) Diverse GPCRs exhibit conserved water networks for stabilization and activation. Proc Natl Acad Sci USA 116(8):3288–3293. https://doi.org/10.1073/pnas.1809251116

[129] Zendman AJ, Cornelissen IM, Weidle UH, Ruiter DJ, van Muijen GN. TM7XN1, a novel human EGF-TM7-like cDNA, detected with mRNA differential display using human melanoma cell lines with different metastatic potential. FEBS Lett. 1999;446:292–298

Appendix A1 - Online Script for Extracting various descriptors

```
import pandas as pd
from propy.GetProteinFromUniprot import GetProteinSequence as gps
from propy.PyPro import GetProDes

data = pd.read_csv('Final_correected_GPCR_TransNOnGPCR.csv')
protein_id = data['ID']
protein_id_list = list(protein_id)

result = []
protein_id_error_index=[]
for idx,pro_id in enumerate(protein_id_list):
print(idx)
protein_sequence = gps(pro_id)
descriptor = GetProDes(protein_sequence)
try:
PAAC= descriptor.GetPAAC()
#SOCN=descriptor.GetSOCN()
#PAAC = descriptor.GetPAAC(lamda=5,weight=0.1)
except Exception as e:
# raise e
print('Error occurred finding the protein
                         descriptors for '+pro_id)
protein_id_error_index.append(idx)
continue
result.append(PAAC)
protein_id = protein_id.drop(labels=protein_id_error_index).
reset_index()
```

```
pd.concat([protein_id ,pd.DataFrame(result)], axis=1).
drop(labels=['index'], axis=1).to_csv('GPCR_TRANS_PAAC.csv')
pd.DataFrame({'Protein_id_error': protein_id_error_index}).
to_csv('GPCR_TRANS_PAACerror1.csv')
```

Appendix A2 - Offline Script for Extracting various descriptors

```
import sys
from propy.PyPro import GetProDes
import pandas as pd
import time
start_time = time.time()

result = []
protein_id_error_index =[]
protein_id = []
# number_of_iteration = 5
# Total_time = []


with open('uniprot_201_300.fasta', 'r') as ProSeqfile:
text = ProSeqfile.read()
#def sequence_extraction(text):
textlist = text.split('>')
for idx,ls in enumerate(textlist):
sub = ls.split("\n", 1)
if idx==0:
continue
```

```
pro_id = sub[0]
#pro_id = sub[0].split("|")[1]
protein_id.append(pro_id)
protein_sequence = ''.join(sub[1].split("\n"))
descriptor = GetProDes(protein_sequence)
try:
ALL= descriptor.GetReduced()
except Exception as e:
# raise e
print('Error occurred finding the protein descriptors for '+pro_id)
protein_id_error_index.append(idx)
continue
result.append(ALL)
#print(pro_id, ':',ALL)

protein_id = pd.Series(protein_id)
protein_id = protein_id.drop(labels=protein_id_error_index)
.reset_index()
#pd.concat([protein_id,pd.DataFrame(result)],axis=1)
.drop(labels=['index'],
axis=1).to_csv('uniprot_TAS2R.csv')
df1= pd.concat([protein_id,pd.DataFrame(result)],axis=1).
drop(labels=['index'],
axis=1)
array = df1.values
X_test = array[:,1:]
```

Appendix B - UFS, PCA & ISA

```
import pandas as pd
import numpy as np
import random as rand
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from random import choice, seed
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.feature_selection import chi2

##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)

np.random.seed(75)
dataset = pd.read_csv('GSE4115.csv', header=None, skiprows=1)
 # The main dataset to be split into training and test
#X = dataset.iloc[:,:-1]
#1165 446 1357

array = dataset.values
X = array[:,1:-1]
Y = array[:,-1]
#X = dataset.iloc[:,1:1356]
#y = dataset.iloc[:,-1]

# training_set,test_set,training_labels,test_labels =
                        train_test_split(X,Y ,test_size=0.3,
#
random_state=42)
# #print(test_set)
# # Saving the trainingset and testset
# pd.concat([training_set, training_labels], axis=1).to_csv
                        ('Training_Set.csv', header=0)
# pd.concat([test_set, test_labels], axis=1).
to_csv('Test_Set.csv', header=0)

# Feature Extraction with Univariate Statistical Tests
(Chi-squared for classification)

# load data

# array = dataframe.values
# X = array[:,0:8]
# Y = array[:,8]
# feature extraction

test = SelectKBest(score_func=f_classif, k=700)
fit = test.fit(X, Y)
# summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
#ft=np.concatenate((features,Y, axis=1)
# summarize selected features
print(features[0:5,:])
```

```
np.savetxt('features_set.csv', features,'%s',',')

#randomly generate the rows to pick out for the trainingset
and the rest are used for the testset
training_set,test_set,training_labels,test_labels =
train_test_split(features,Y ,test_size=0.3,
random_state=75)
# #print(test_set)
# # Saving the trainingset and testset
training_labels = np.reshape(training_labels,
(training_labels.shape[0],1))
test_labels = np.reshape(test_labels,(test_labels.shape[0],1))
np.savetxt('Training_Set.csv',np.concatenate(
(training_set,training_labels),axis=1),'%s',',')
 # the first column of this file contains the indices
 showing how random the rows were selected
np.savetxt('Test_Set.csv',np.concatenate(
(test_set,test_labels),axis=1),
'%s',',')
# convert the two datasets(dataframe) into numpy arrays
testArrays = test_set
trainingArrays = training_set

#feature scaling
sc=StandardScaler()
trainingArrays=sc.fit_transform(trainingArrays)
#print(trainingArrays)
testArrays=sc.transform(testArrays)
```

```
#applying pca
pca = PCA(n_components =95)
trainingArrays = pca.fit_transform(trainingArrays)
#print(trainingArrays)
testArrays = pca.transform(testArrays)
explained_variance = pca.explained_variance_ratio_
#print(explained_variance)

predicted_class_list = []

for index, testRow in enumerate(testArrays):
# keeps track of the similarity scores for each row
(the row in the test dataset)
similarity_score_list = []
for row in trainingArrays:
norm = np.linalg  # class to access norm-2

# calculates the similarity score
simi_score = np.divide(np.dot(testRow, row),
np.multiply(norm.norm(testRow), norm.norm(row)))
similarity_score_list.append(simi_score)

index_maxvalue = similarity_score_list.
index(max(similarity_score_list))
class_maxvalue = training_labels[index_maxvalue]
 # gets the class of the maximum similarity score
predicted_class_list.append(class_maxvalue)
```

```
# print('\nThe test protein[',index+1,']
belongs to :',class_maxvalue)

# This adds the predicted classes to the last
#column in the testArrays
testArrays = np.concatenate((testArrays, test_labels,
np.array(predicted_class_list)), axis=1)

# Checking the accuracy
true_positives = 0  # variable to hold the
number of occurances where
 actual is equal to predicted
pred_classes_count_dict = {}
test_classes_count_dict = {}
for i in range(len(test_labels)):
testLabel = str(test_labels[i]).strip().lower()
# count the classes in test
if testLabel in test_classes_count_dict.keys():
test_classes_count_dict[testLabel] += 1
else:
test_classes_count_dict[testLabel] = 1

if testLabel == str(predicted_class_list[i]).strip().lower():
true_positives += 1
# count the predicted classes
if testLabel in pred_classes_count_dict.keys():
pred_classes_count_dict[testLabel] += 1
```

```
else:
pred_classes_count_dict[testLabel] = 1

for key, value in pred_classes_count_dict.items():
print('The Accuracy of class [', key, '], is :',
pred_classes_count_dict[key] /
 test_classes_count_dict[key] * 100)
print('The Overall Accuracy is :', true_positives /
len(testArrays) * 100)

# save the predictions made to a file
np.savetxt('Test_Set_predicted_VI.csv', testArrays, '%s', ',')
```

Appendix C - PCA & ISA

```
''' After successful running of this code the accuracy is printed
 to the console
and three files named as:
1. 'Training_Set.csv'
2. 'Test_Set.csv'
3. 'Test_Set_predicted.csv'  are produced'''

import pandas as pd
import numpy as np
import random as rand
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from random import choice, seed

np.random.seed(75)
dataset = pd.read_csv('GSE4115.csv', header=0)
 # The main dataset to be split into training and test
#X = dataset.iloc[:,:-1]
#1165 446 1357
X = dataset.iloc[:,:-1]
y = dataset.iloc[:,-1]

#randomly generate the rows to pick out for the training set
and the rest are used for the testset
training_set,test_set,training_labels,test_labels =
 train_test_split(X,y ,test_size=0.3,
random_state=75)

np.savetxt('Test_Set_check.csv', test_set,'%s',',')
#print(test_set)
# Saving the trainingset and testset
pd.concat([training_set,training_labels],axis=1).to_csv
('Training_Set.csv',header=0) # the first column of this file
 contains the indices showing how random the rows were selected
pd.concat([test_set,test_labels],axis=1).
to_csv('Test_Set.csv',header=0)

# convert the two datasets(dataframe) into numpy arrays
testArrays = test_set.iloc[:,1:].values
trainingArrays = training_set.iloc[:,1:].values

#feature scaling
sc=StandardScaler()
trainingArrays=sc.fit_transform(trainingArrays)
#print(trainingArrays)
testArrays=sc.transform(testArrays)

#applying pca
pca = PCA(n_components=400)
trainingArrays = pca.fit_transform(trainingArrays)
#print(trainingArrays)
testArrays = pca.transform(testArrays)
explained_variance = pca.explained_variance_ratio_
```

```
#print(explained_variance)


# this list holds the predicted classes for the test proteins
predicted_class_list=[]

for index,testRow in enumerate(testArrays):
# keeps track of the similarity scores for each row(the row in
 the test dataset)
similarity_score_list = []
for row in trainingArrays:
norm = np.linalg # class to access norm-2

# calculates the similarity score
simi_score = np.divide(np.dot(testRow,row),
np.multiply(norm.norm(testRow),norm.norm(row)))
similarity_score_list.append(simi_score)



index_maxvalue = similarity_score_list.
index(max(similarity_score_list))
class_maxvalue = training_labels.iloc[index_maxvalue]
# gets the class of the maximum similarity score
predicted_class_list.append(class_maxvalue)
```

```
# This adds the predicted classes to the last
# column in the testArrays
test_set_row = test_set.iloc[:,0].shape[0]
testArrays = np.concatenate((testArrays,test_labels.
values.reshape
([len(test_labels),1]),
np.array([predicted_class_list]).T,test_set.iloc[:,0].
values.reshape
([test_set_row,1])),axis=1)

# Checking the accuracy
true_positives = 0 # variable to hold the number of occurances
where actual is equal to predicted
pred_classes_count_dict = {}
test_classes_count_dict = {}
for i in range(len(test_labels)):
testLabel = str(test_labels.iloc[i]).strip().lower()
# count the classes in test added str in line 119 and 112
if testLabel in test_classes_count_dict.keys():
test_classes_count_dict[testLabel]+=1
else:
test_classes_count_dict[testLabel]=1

if testLabel== str(predicted_class_list[i]).strip().lower():
true_positives +=1
# count the predicted classes
```

```
if testLabel in pred_classes_count_dict.keys():
pred_classes_count_dict[testLabel]+=1
else:
pred_classes_count_dict[testLabel]=1

for key,value in pred_classes_count_dict.items():
print('The Accuracy of class [',key,'], is :',
pred_classes_count_dict[key]/test_classes_count_dict[key]*100)
print('The Overall Accuracy is :',
true_positives/len(testArrays)*100)


#save the predictions made to a file
np.savetxt('Test_Set_predicted_simi.csv', testArrays,'%s',',')
```

Appendix D- PCA & Random Forest

```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#import the data set

np.random.seed(75)

dataset=pd.read_csv('GPCR_TransNOnGPCR.csv',header=0)
X=dataset.iloc[:,1:-1].values
y=dataset.iloc[:,-1].values

#splitting the dataset into the Training set and test set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train,y_test=
train_test_split(X,y ,test_size=0.3,random_state=75)

#feature scaling

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
#print(X_train)
X_test=sc.transform(X_test)
```

```
#applying pca
from sklearn.decomposition import PCA
pca = PCA(n_components = 75)
X_train = pca.fit_transform(X_train)
print(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)

# Fitting RandomForst Classifier to the Training set
from sklearn.ensemble import RandomForestClassifier as RFC
classifier = RFC(n_estimators=1000,criterion='entropy',75)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Appendix D- Logistic regression - Model Testing (Binary)

```
from sklearn.metrics import classification_report ,
confusion_matrix accuracy_score ,roc_curve , roc_auc_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import statistics as st
from sklearn.feature_selection import SelectKBest ,
f_classif ,SelectFpr
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.metrics import matthews_corrcoef


 # The main dataset to be split into training and test
dataset = pd.read_csv('GPCRdb_GPCR_NonGPCR.csv', header=0)


#dataset1 = pd.read_csv('Trans_id.csv',header=0)
dataset.head()
```

```
array = dataset.values
X = array[:,1:-1]
print(X.shape)
Y = array[:,-1]

test = SelectKBest(score_func=f_classif, k=795)
#test = SelectFpr(score_func=f_classif,alpha=0.01)
fit = test.fit(X, Y)
features = fit.transform(X)
#print(features.shape)
# # Et = ExtraTreesClassifier(n_estimators=60)
# # Et = Et.fit(X, Y)
# # model = SelectFromModel(Et, prefit=True)
# # X_new = model.transform(X)
#
# clf = ExtraTreesClassifier(n_estimators=750)
# clf = clf.fit(X, Y)
# model = SelectFromModel(clf, prefit=True)
# X_new = model.transform(X)


number_of_iteration = 5
overall_accuracies = []
overall_TmFPR_mean=[]
overall_NTmFPR_mean=[]
for iter_point in range(number_of_iteration):
        print("\nIteration point: %d" % iter_point)
```

```
X_train, X_test, y_train, y_test =
  train_test_split(features, Y, test_size=0.3)



#print(y_test.shape())
X_train = X_train[:, 0:]
Xtest_ids = list(set(X_test[:, 0]))
X_test_ids = X_test[:, 0].reshape(X_test.shape[0], 1)
X_test = X_test[:, 0:]

# # Applying synthetic oversampling technique
# sm = SMOTE()
# X_res, y_res = sm.fit_resample(X_train, y_train)
# from collections import Counter
#
# print(sorted(Counter(y_res).items()))
# print(len(y_res))

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Applying synthetic oversampling technique
sm = SMOTE(random_state=0)
X_res, y_res = sm.fit_resample(np.asarray(X_train),
                   np.asarray(y_train))
from collections import Counter
```

```
print(sorted(Counter(y_res).items()))
print(len(y_res))




clf = LogisticRegression(multi_class="multinomial",
      penalty='l2', tol=0.1, solver='newton-cg')

clf1 = clf.fit(X_train, y_train)
y_pred = clf1.predict(X_test)

### When determining the false positive rates

y_test = np.reshape(y_test, (y_test.shape[0], 1))
y_pred = np.reshape(y_pred, (y_pred.shape[0], 1))
print(X_test.shape)
print(y_test.shape)
print(y_pred.shape)
testArrays = np.concatenate(
          (X_test_ids, y_test, y_pred), axis=1)
All_Non_GPCRs = testArrays[testArrays[:, -2]
                                          == 'NonGPCR']
# extracts the last but 2 columns from testarray set
All_Non_GPCRs = All_Non_GPCRs[:, 0]
All_Non_GPCRs_IDs = list(set(All_Non_GPCRs))
Number_of_Nongpcr = len(All_Non_GPCRs_IDs)
```

```
False_Non_GPCRs = testArrays[(testArrays[:, -2]
  == 'NonGPCR')\& (testArrays[:, -1] == 'GPCR')]
False_Non_GPCRs_ids = False_Non_GPCRs[:, 0]
dataset_ids = list(set(False_Non_GPCRs_ids))
# print('dataset_ids:', dataset_ids)
# print(dataset_ids)
dataset1_ids = dataset1.iloc[:, 0].tolist()
# # print('datase1_ids:',dataset1_ids)
#
similar_ids = list(set(dataset_ids)
                        \& set(dataset1_ids))
similar_ids_2 = list(set(Xtest_ids)
                        \& set(dataset1_ids))
Numer_Tm = len(similar_ids)
Numer_NTm = len(dataset_ids) - Numer_Tm

denom_Tm = len(similar_ids_2)
denom_NTm = Number_of_Nongpcr - denom_Tm
#print('False positive rate among Tms= ',
                        Numer_Tm / denom_Tm)
overall_TmFPR_mean.append(Numer_Tm / denom_Tm)
overall_NTmFPR_mean.append(Numer_NTm / denom_NTm)
overall_TmFPR_mean.append(Numer_Tm / denom_Tm)
overall_NTmFPR_mean.append(Numer_NTm / denom_NTm)
np.savetxt('LogisticTest_Arrays_%d.csv' % iter_point,
                testArrays, '%s', ',')

# #Computing false and true positive rates
```

```
        matt_corr=matthews_corrcoef(y_test, y_pred)
        print("matt_corr= ",matt_corr)
        print(confusion_matrix(y_test,y_pred))
        print(classification_report(y_test,y_pred))
        print(accuracy_score(y_test, y_pred))
        overall_accuracies.append(accuracy_score
                                        (y_test, y_pred))

print("\nThe average accuracy  and standard deviation after %d
                    iteration are %f and %f" %(number_of_iteration,
sum(overall_accuracies)/number_of_iteration,
                         st.stdev(overall_accuracies)))
```

Appendix E- Logistic regression - Model Testing (Multiclass)

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score,roc_curve, roc_auc_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import statistics as st
from sklearn.feature_selection import SelectKBest,
                     f_classif,SelectFpr
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.metrics import matthews_corrcoef



dataset = pd.read_csv('GPCRdb_GPCR_NonGPCR.csv', header=0)


#dataset1 = pd.read_csv('Trans_id.csv',header=0)
dataset.head()
```

```
array = dataset.values
X = array[:,1:-1]
print(X.shape)
Y = array[:,-1]

test = SelectKBest(score_func=f_classif, k=795)
#test = SelectFpr(score_func=f_classif,alpha=0.01)
fit = test.fit(X, Y)
features = fit.transform(X)
#print(features.shape)



number_of_iteration = 5
overall_accuracies = []
overall_TmFPR_mean=[]
overall_NTmFPR_mean=[]
for iter_point in range(number_of_iteration):
        print("\nIteration point: %d" % iter_point)

        X_train, X_test, y_train, y_test = train_test_split
                            (features, Y, test_size=0.3)


        #print(y_test.shape())
        X_train = X_train[:, 0:]
        Xtest_ids = list(set(X_test[:, 0]))
        X_test_ids = X_test[:, 0].reshape(X_test.shape[0], 1)
```

```
X_test = X_test[:, 0:]



# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Applying synthetic oversampling technique
sm = SMOTE(random_state=0)
X_res, y_res = sm.fit_resample(np.asarray(X_train),
                                       np.asarray(y_train))
from collections import Counter

print(sorted(Counter(y_res).items()))
print(len(y_res))




clf = LogisticRegression(multi_class="multinomial",
          penalty='l2', tol=0.1, solver='newton-cg')
#liblinear, saga,newton-cg, lbfgs, sag and saga
#clf1 = rfe.fit(X_train, y_train)
#clf1 = clf.fit(X_res, y_res)
clf1 = clf.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
```

```
        ### When determining the false positive rates


        # #Computing false and true positive rates


        matt_corr=matthews_corrcoef(y_test, y_pred)
        print("matt_corr= ",matt_corr)
        print(confusion_matrix(y_test,y_pred))
        print(classification_report(y_test,y_pred))
        print(accuracy_score(y_test, y_pred))
        overall_accuracies.append(accuracy_score(
                                          y_test, y_pred))

print("\nThe average accuracy  and standard deviation after %d
              iteration are %f and %f" %(number_of_iteration,
sum(overall_accuracies)/number_of_iteration,
                              st.stdev(overall_accuracies)))
```

Appendix F- Logistic regression - Web-server design testing

```
import sys
from propy.PyPro import GetProDes
import pandas as pd
import time
start_time = time.time()

result = []
protein_id_error_index =[]
protein_id = []
# number_of_iteration = 5
# Total_time = []



with open('uniprot_100000.fasta', 'r') as ProSeqfile:
        text = ProSeqfile.read()
        #def sequence_extraction(text):
        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        pro_id = sub[0]
        #pro_id = sub[0].split("|")[1]
        protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        descriptor = GetProDes(protein_sequence)
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
        # raise e
        print('Error occurred finding the protein
                        descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)
#print(pro_id, ':', ALL)

protein_id = pd.Series(protein_id)
protein_id = protein_id.drop(labels=
                    protein_id_error_index).reset_index()
#pd.concat([protein_id,pd.DataFrame(result)],axis=1).drop(
          labels=['index'],axis=1).to_csv('uniprot_TAS2R.csv')
df1= pd.concat([protein_id,pd.DataFrame(result)],axis=1).drop(
                            labels=['index'],axis=1)
array = df1.values
X_test = array[:,1:]
#print("--- %s seconds ---" % (time.time() - start_time))



#pd.DataFrame({'Protein_id_error':protein_id_error_index}).
                    to_csv('propy error_uniprot_TAS2R.csv')

import pandas as pd
# # df = pd.read_csv('wine.data.csv', header=None)
```

```
from sklearn.metrics import classification_report,
            confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
#import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import statistics as st
from sklearn.feature_selection import SelectKBest,
        f_classif,SelectFpr,chi2,SelectFdr,SelectFwe
import numpy as np
from sklearn.metrics import jaccard_similarity_score
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import RFE


#UFS_PCA_ISA

##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)

df = pd.read_csv('GPCRPenDb_All.csv', header=0)

array = df.values
X_train = array[:,1:-1]
```

```
#X_imp=X[:,imp_feat]
#print(X_imp)
#print(X_imp)
Y = array[:,-1]



 sc=StandardScaler()
 trainingArrays=sc.fit_transform(X_train)
#print(trainingArrays)
 X_test =sc.transform(X_test)


#print(X)

# Feature Scaling
 sc = StandardScaler()
#X = sc.fit_transform(X_test)

 model = LogisticRegression(multi_class="multinomial",
                            penalty='l2', tol=0.1, solver='lbfgs')
#liblinear, saga,newton-cg, lbfgs, sag and saga
 model = model.fit(trainingArrays, Y)
 predictions = model.predict(X_test)
 df_list = pd.Series(predictions)
#df_list.to_csv('results_adaboost.csv')
#df_list.to_csv('results_logistic_ lbfgs.csv')
```

```
print(predictions)
#df_list =pd.Series(predictions)
#df_list.to_csv('results_pca-isa.csv')
#print(df_list)
for seq in predictions:
if seq == 'CNT':
print('It is a non-GPCR but',seq,'\n \n')
elif seq == 'CN':
print('It is a non-GPCR but', seq, '\n \n')
elif seq[0] == 'G':
f = seq.split('_')
print('It is a {0} protein and belongs to;
 \n family: {1} \n subfamily:  {2} \n subsubfamily: {3}  \n
 subtype: {4} \n \n'.format(f[0],f[1],f[2],f[3],f[4]))

#final_time = time.time() - start_time
#print(final_time)
#Total_time.append(final_time)
print("--- %s seconds ---" % (time.time() - start_time))
```

Appendix G- Logistic regression - Web-server final Algorithm

```
def Log–reg(filename):
import sys
from propy.PyPro import GetProDes
import pandas as pd
from Config import DB_ROOT



result = []
protein_id_error_index =[]
protein_id = []

with open(filename, 'r') as ProSeqfile:
        text = ProSeqfile.read()

        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        pro_id = sub[0]
        protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        print(protein_sequence)
        descriptor = GetProDes(protein_sequence.upper())
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
        # raise e
        print('Error occurred finding the protein
                        descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)

df1 = pd.DataFrame(result)
array = df1.values
X_test = array[:, :]

from sklearn.metrics import classification_report,
                        confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                    confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statistics as st

# UFS_PCA_ISA

##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)
```

```
df = pd.read_csv('GPCRPenDb_All.csv', header=0)

array = df.values
X_train = array[:, 1:-1]

# X_imp=X[:,imp_feat]
# print(X_imp)
# print(X_imp)
Y = array[:, -1]

sc = StandardScaler()
trainingArrays = sc.fit_transform(X_train)
# print(trainingArrays)
X_test = sc.transform(X_test)

# print(X)

# Feature Scaling
sc = StandardScaler()
# X = sc.fit_transform(X_test)




  clf= LogisticRegression(penalty='l2', tol=0.01, solver='saga')

model = model.fit(trainingArrays, Y)
```

```
predictions = model.predict(X_test)

print(predictions)

df_list =pd.Series(predictions)

for seq in predictions:
        if seq == 'CNT':
        #label1.append('{} is a non-GPCR but {}
                                \n \n'.format(id,seq))
        return '-\t-\t-\t-'
        #print('It is a non-GPCR but',seq,'\n \n')
        elif seq == 'CN':
        # label2.append('{} is a non-GPCR but {}
                                  \n \n'.format(id,seq))
        return '-\t-\t-\t-'
        # print('It is a non-GPCR but', seq, '\n \n')
        elif seq[0] == 'G':
        f = seq.split('_')

        return '\t'.join(f[1:5])

        else:
        return '-\t-\t-\t-'
```

Appendix H- Multi-layer perceptron - Model testing (Binary)

```
#
#

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                  confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statistics as st
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest,
        f_classif,SelectFpr,chi2,SelectFdr,SelectFwe



#
df = pd.read_csv('gpcrpendb_GPCR_NG.csv',header=0)
dataset1 = pd.read_csv('Trans_id.csv',header=0)
#ff = feature_Selection(df.iloc[:,1:])
#print(list(ff))
#print(feature_ranking(ff))
```

```
#imp_feat=feature_ranking(ff)[0:650]
# imp_feat = np.array(imp_feat)+1
# imp_feat = np.insert(imp_feat,0,0)
 array = df.values
 df.head()
 array = df.values
X = array[:,1:-1]
#X_imp=X[:,imp_feat]
Y = array[:,-1]

 test = SelectKBest(score_func=f_classif, k=7)
# #test = SelectFpr(score_func=f_classif,alpha=0.001)
 ft = test.fit(X, Y)
 features = ft.transform(X)

# clf = ExtraTreesClassifier(n_estimators=900)
# clf = clf.fit(X, Y)
# model = SelectFromModel(clf, prefit=True)
# X_new = model.transform(X)
#print(X_new.shape)



 number_of_iteration = 5
 overall_accuracies = []
 overall_TmFPR_mean=[]
 overall_NTmFPR_mean=[]
 for iter_point in range(number_of_iteration):
         print("\nIteration point: %d" % iter_point)
```

```
X_train, X_test, y_train, y_test =
            train_test_split(X, Y, test_size=0.3)
X_train = X_train[:, 1:]
#Xtest_ids = list(set(X_test[:, 0]))
X_test_ids = X_test[:, 0].reshape(X_test.shape[0], 1)
X_test = X_test[:, 1:]

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# #applying pca
from sklearn.decomposition import PCA

# pca = PCA(n_components=800)
# X_train= pca.fit_transform(X_train)
# # print(trainingArrays)
# X_test = pca.transform(X_test)
# explained_variance = pca.explained_variance_ratio_
# print(explained_variance.sum())



clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(15,))
#rfe = RFE(clf, 500)lbfgs
# clf= LogisticRegression(penalty='l2',
                tol=0.01, solver='saga')lbfgs sgd, adam,
#clf1 = rfe.fit(X_train, y_train)
clf1=clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_test = np.reshape(y_test, (y_test.shape[0], 1))
y_pred = np.reshape(y_pred, (y_pred.shape[0], 1))
# print(X_test.shape)
# print(y_test.shape)
# print(y_pred.shape)

testArrays = np.concatenate((X_test_ids,
                            y_test, y_pred), axis=1)
All_Non_GPCRs = testArrays[testArrays[:, -2] ==
                                            'NonGPCR']
All_Non_GPCRs = All_Non_GPCRs[:, 0]
All_Non_GPCRs_IDs = list(set(All_Non_GPCRs))
Number_of_Nongpcr = len(All_Non_GPCRs_IDs)
False_Non_GPCRs = testArrays[(testArrays[:, -2] ==
        'NonGPCR') & (testArrays[:, -1] == 'GPCR')]
False_Non_GPCRs_ids = False_Non_GPCRs[:, 0]

dataset_ids = list(set(False_Non_GPCRs_ids))
# print('dataset_ids:', dataset_ids)
# print(dataset_ids)
dataset1_ids = dataset1.iloc[:, 0].tolist()
# print('datase1_ids:',dataset1_ids)

similar_ids = list(set(dataset_ids)
                        & set(dataset1_ids))
        similar_ids_2 = list(set(Xtest_ids)
                              & set(dataset1_ids))
        Numer_Tm = len(similar_ids)
        Numer_NTm = len(dataset_ids) - Numer_Tm
        denom_Tm = len(similar_ids_2)
        denom_NTm = Number_of_Nongpcr - denom_Tm
        print('False positive rate among Tms= ',
                              Numer_Tm / denom_Tm)
        overall_TmFPR_mean.append(Numer_Tm / denom_Tm)
        overall_NTmFPR_mean.append(Numer_NTm / denom_NTm)
        overall_TmFPR_mean.append(Numer_Tm / denom_Tm)
        overall_NTmFPR_mean.append(Numer_NTm / denom_NTm)
        np.savetxt('LogisticTest_Arrays_%d.csv'
                  % iter_point, testArrays, '%s', ',')

        print(confusion_matrix(y_test, y_pred))
        print(classification_report(y_test, y_pred))
        print(accuracy_score(y_test, y_pred))
        overall_accuracies.append(accuracy_score
                                  (y_test, y_pred))
        #np.savetxt('NeuralNetTest_Arrays_%d.csv'
                 % iter_point, testArrays, '%s', ',')


print("\nThe average accuracy  and standard deviation after
    %d iteration are %f and %f" %(number_of_iteration,
    st.mean(overall_accuracies),st.stdev(overall_accuracies)))
```

Appendix I- Multi-layer perceptron - Model testing (Multi-class)

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                 confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statistics as st
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest,
        f_classif,SelectFpr,chi2,SelectFdr,SelectFwe


#
df = pd.read_csv('gpcrpendb.csv', header=0)
dataset1 = pd.read_csv('Trans_id.csv',header=0)
#ff = feature_Selection(df.iloc[:,1:])
#print(list(ff))
#print(feature_ranking(ff))
#imp_feat=feature_ranking(ff)[0:650]
# imp_feat = np.array(imp_feat)+1
# imp_feat = np.insert(imp_feat,0,0)
```

```
array = df.values
df.head()
array = df.values
X = array[:,1:-1]
#X_imp=X[:,imp_feat]
Y = array[:,-1]

test = SelectKBest(score_func=f_classif, k=7)
# #test = SelectFpr(score_func=f_classif,alpha=0.001)
ft = test.fit(X, Y)
features = ft.transform(X)

# clf = ExtraTreesClassifier(n_estimators=900)
# clf = clf.fit(X, Y)
# model = SelectFromModel(clf, prefit=True)
# X_new = model.transform(X)
#print(X_new.shape)


number_of_iteration = 5
overall_accuracies = []
overall_TmFPR_mean=[]
overall_NTmFPR_mean=[]
for iter_point in range(number_of_iteration):
        print("\nIteration point: %d" % iter_point)

        X_train, X_test, y_train, y_test =
                    train_test_split(X, Y, test_size=0.3)
```

```
X_train = X_train[:, 1:]
#Xtest_ids = list(set(X_test[:, 0]))
X_test_ids = X_test[:, 0].reshape(X_test.shape[0], 1)
X_test = X_test[:, 1:]

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# #applying pca
from sklearn.decomposition import PCA




clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(15,))
#rfe = RFE(clf, 500)lbfgs
# clf= LogisticRegression(penalty='l2', tol=0.01,
                              solver='saga')lbfgs sgd, adam,
#clf1 = rfe.fit(X_train, y_train)
clf1=clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_test = np.reshape(y_test, (y_test.shape[0], 1))
y_pred = np.reshape(y_pred, (y_pred.shape[0], 1))
# print(X_test.shape)
# print(y_test.shape)
# print(y_pred.shape)
```

```
        print(confusion_matrix(y_test, y_pred))
        print(classification_report(y_test, y_pred))
        print(accuracy_score(y_test, y_pred))
        overall_accuracies.append(accuracy_score
                                          (y_test, y_pred))
        #np.savetxt('NeuralNetTest_Arrays_%d.csv'
                      % iter_point, testArrays, '%s', ',')


print("\nThe average accuracy  and standard deviation after
            %d iteration are %f and %f" %(number_of_iteration,
st.mean(overall_accuracies),st.stdev(overall_accuracies)))
```

Appendix J- Multi-layer perceptron - Web-server testing

```
import sys
from propy.PyPro import GetProDes
import pandas as pd
import time
start_time = time.time()

#uniprot25
result = []
protein_id_error_index =[]
protein_id = []
with open('uniprot_20000.fasta', 'r') as ProSeqfile:
        text = ProSeqfile.read()
        #def sequence_extraction(text):
        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        pro_id = sub[0]
        #pro_id = sub[0].split("|")[1]
        protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        descriptor = GetProDes(protein_sequence)
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
        # raise e
        print('Error occurred finding the protein
                                    descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)
#print(pro_id, ':', ALL)

protein_id = pd.Series(protein_id)
protein_id = protein_id.drop(labels=protein_id_error_index)
                                                .reset_index()

df1=pd.concat([protein_id,pd.DataFrame(result)],axis=1)
                                    .drop(labels=['index'],axis=1)
array = df1.values
X_test = array[:,1:]






import pandas as pd
# # df = pd.read_csv('wine.data.csv', header=None)

from sklearn.metrics import classification_report,
                confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                    confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statistics as st

#UFS_PCA_ISA

##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)

#GPCRPenDb_All_gpcr_nongpcr_DPC_QSO_Geary_AAC
#GPCRPenDb_All_gpcr_nongpcr_final training set
df = pd.read_csv('GPCRPenDb_All.csv', header=0)
# ff = feature_Selection(df.iloc[:,1:])
# # #print(list(ff))
# # #print(feature_ranking(ff))
# imp_feat=feature_ranking(ff)[0:990]
# imp_feat = np.array(imp_feat)+1
# imp_feat = np.insert(imp_feat,0,0)
array = df.values
X_train = array[:,1:-1]


#X_imp=X[:,imp_feat]
#print(X_imp)
```

```
#print(X_imp)
Y = array[:,-1]



 sc=StandardScaler()
 trainingArrays=sc.fit_transform(X_train)
#print(trainingArrays)
 X_test =sc.transform(X_test)


#print(X)

# Feature Scaling
 sc = StandardScaler()
#X = sc.fit_transform(X_test)

 model = MLPClassifier(solver='lbfgs', alpha=1e-5,
 hidden_layer_sizes=(15,))
# rfe = RFE(clf, 500)lbfgs sgd, adam,
# clf= LogisticRegression(penalty='l2', tol=0.01, solver='saga')
# clf1 = rfe.fit(X_train, y_train)
#liblinear, saga,newton-cg, lbfgs, sag and saga
 model = model.fit(trainingArrays, Y)
 predictions = model.predict(X_test)
 df_list =pd.Series(predictions)
#df_list.to_csv('results_adaboost.csv')
#df_list.to_csv('results_logistic_ lbfgs.csv')
```

```
print(predictions)
#print(classification_report(test_labels, predicted_class_list))
#df_list =pd.Series(predictions)
#df_list.to_csv('results_pca-isa.csv')
#print(df_list)
for seq in predictions:
        if seq == 'CNT':
        print('It is a non-GPCR but',seq,'\n \n')
        elif seq == 'CN':
        print('It is a non-GPCR but', seq, '\n \n')
        elif seq[0] == 'G':
        f = seq.split('_')
    print('It is a {0} protein and belongs to; \n family: {1}
    \n subfamily:  {2} \n subsubfamily: {3}  \n subtype: {4}
    \n \n'.format(f[0],f[1],f[2],f[3],f[4]))

print("--- %s seconds ---" % (time.time() - start_time))
```

Appendix K- Multi-layer perceptron - Web-server Final Algorithm

```
def MLP_NN(filename):
import sys
from propy.PyPro import GetProDes
import pandas as pd
from Config import DB_ROOT



result = []
protein_id_error_index =[]
protein_id = []

with open(filename, 'r') as ProSeqfile:
        text = ProSeqfile.read()

        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        pro_id = sub[0]
        protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        print(protein_sequence)
        descriptor = GetProDes(protein_sequence.upper())
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
```

```
        # raise e
        print('Error occurred finding the protein
                              descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)

df1 = pd.DataFrame(result)
array = df1.values
X_test = array[:, :]

from sklearn.metrics import classification_report,
                     confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                          confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import statistics as st

# UFS_PCA_ISA

##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)

df = pd.read_csv('GPCRPenDb_All.csv', header=0)
```

```
# ff = feature_Selection(df.iloc[:,1:])
# # #print(list(ff))
# # #print(feature_ranking(ff))
# imp_feat=feature_ranking(ff)[0:990]
# imp_feat = np.array(imp_feat)+1
# imp_feat = np.insert(imp_feat,0,0)
array = df.values
X_train = array[:, 1:-1]

# X_imp=X[:,imp_feat]
# print(X_imp)
# print(X_imp)
Y = array[:, -1]

sc = StandardScaler()
trainingArrays = sc.fit_transform(X_train)
# print(trainingArrays)
X_test = sc.transform(X_test)

# print(X)

# Feature Scaling
sc = StandardScaler()
# X = sc.fit_transform(X_test)

from sklearn.decomposition import PCA
pca = PCA(n_components=500)
trainingArrays = pca.fit_transform(trainingArrays)
```

```
#print(trainingArrays)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print(explained_variance.sum())

model = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(15,))
# rfe = RFE(clf, 500)lbfgs sgd, adam,
# clf= LogisticRegression(penalty='l2', tol=0.01, solver='saga')
# clf1 = rfe.fit(X_train, y_train)
# liblinear, saga,newton-cg, lbfgs, sag and saga
model = model.fit(trainingArrays, Y)
predictions = model.predict(X_test)

print(predictions)
#print(classification_report(test_labels, predicted_class_list))
df_list =pd.Series(predictions)

for seq in predictions:
        if seq == 'CNT':
        #label1.append('{} is a non-GPCR but {}
                                \n \n'.format(id,seq))
        return '-\t-\t-\t-'
        #print('It is a non-GPCR but',seq,'\n \n')
        elif seq == 'CN':
        # label2.append('{} is a non-GPCR but {}
                              \n \n'.format(id,seq))
        return '-\t-\t-\t-'
```

```
# print('It is a non-GPCR but', seq, '\n \n')
elif seq[0] == 'G':
f = seq.split('_')

return '\t'.join(f[1:5])

else:
return '-\t-\t-\t-'
```

Appendix L- PCA-ISA- Web-server Final Algorithm

```
def Pca_Isa(filename):
import sys
from propy.PyPro import GetProDes
import pandas as pd
from Config import DB_ROOT



result = []
protein_id_error_index =[]
protein_id = []

with open(filename, 'r') as ProSeqfile:
        text = ProSeqfile.read()

        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        pro_id = sub[0]
        protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        print(protein_sequence)
        descriptor = GetProDes(protein_sequence.upper())
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
```

```
        # raise e
        print('Error occurred finding the protein
                    descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)

df1 = pd.DataFrame(result)
array = df1.values
X_test = array[:, :]





import pandas as pd

import os
from sklearn.metrics import classification_report,
            confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
#import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
#import statistics as st
import numpy as np
```

```
from sklearn.metrics import jaccard_similarity_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest,
    f_classif,SelectFpr,chi2,SelectFdr,SelectFwe
#UFS_PCA_ISA

print(DB_ROOT)
df = pd.read_csv('%s/GPCRPenDb.csv'%DB_ROOT, header=0)
print('after df')
array = df.values
X_train = array[:,1:-1]



Y = array[:,-1]



sc=StandardScaler()
trainingArrays=sc.fit_transform(X_train)
#print(trainingArrays)
testArrays=sc.transform(X_test)


#NMF
from sklearn.decomposition import NMF
```

```
#applying pca
pca = PCA(n_components=500)
trainingArrays = pca.fit_transform(trainingArrays)
#print(trainingArrays)
testArrays = pca.transform(testArrays)
explained_variance = pca.explained_variance_ratio_
print(explained_variance.sum())

predicted_class_list = []

for index, testRow in enumerate(testArrays):

similarity_score_list = []
for row in trainingArrays:
norm = np.linalg  # class to access norm-2

# calculates the similarity score



simi_score = np.divide(np.dot(testRow, row),
np.multiply(norm.norm(testRow), norm.norm(row)))
#simi_score=jaccard_similarity_score(testRow, row)
similarity_score_list.append(simi_score)

index_maxvalue = similarity_score_list.index(
                              max(similarity_score_list))
class_maxvalue =Y[index_maxvalue]  # gets the class of the
                         maximum similarity score
```

```
predicted_class_list.append(class_maxvalue)



print(predicted_class_list)
#print(classification_report(test_labels, predicted_class_list))
df_list =pd.Series(predicted_class_list)

for seq in predicted_class_list:
        if seq == 'CNT':
        #label1.append('{} is a non-GPCR but {}
                                        \n \n'.format(id,seq))
        return '-\t-\t-\t-'
        #print('It is a non-GPCR but',seq,'\n \n')
        elif seq == 'CN':
        # label2.append('{} is a non-GPCR but {}
                             \n \n'.format(id,seq))
        return '-\t-\t-\t-'
        # print('It is a non-GPCR but', seq, '\n \n')
        elif seq[0] == 'G':
        f = seq.split('_')

        return '\t'.join(f[1:5])

        else:
        return '-\t-\t-\t-'
```

Appendix M- Logistic regression Model Persistent Algorithm

```
\newpage
import pandas  as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.feature_selection import SelectKBest,
                                     f_classif, SelectFpr
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('GPCRPenDb_All.csv', header=0)

array = data.values
X = array[:,1:-1]

y = array[:,-1]



# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X)
#X_test = sc.transform(X_test)



from sklearn.metrics import classification_report,
                          confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,
    precision_recall_curve, auc, roc_auc_score,
    roc_curve, recall_score, classification_report

from imblearn.over_sampling import SMOTE
from sklearn.externals import joblib


#random_state=75



#X_train, X_test, y_train, y_test =
      train_test_split(X, y, test_size=0.3, random_state=0)


model = LogisticRegression(multi_class="multinomial",
                  penalty='l2', tol=0.01, solver='lbfgs')

model.fit(X_train, y)

joblib.dump(model,'model_persist_GPCR_logistic.joblib')

#model = joblib.load('model_persist_GPCR_logistic.joblib')
```

Appendix N- Support Vector Machine (SVM) Algorithm

```
import sys
from propy.PyPro import GetProDes
import pandas as pd



result = []
protein_id_error_index =[]
protein_id = []
with open('uniprot25.fasta', 'r') as ProSeqfile:
        text = ProSeqfile.read()
        #def sequence_extraction(text):
        textlist = text.split('>')
        for idx,ls in enumerate(textlist):
        sub = ls.split("\n", 1)
        if idx==0:
        continue
        #pro_id = sub[0]
        pro_id = sub[0].split("|")[1]
        #protein_id.append(pro_id)
        protein_sequence = ''.join(sub[1].split("\n"))
        descriptor = GetProDes(protein_sequence)
        try:
        ALL= descriptor.GetALL()
        except Exception as e:
        # raise e
        print('Error occurred finding the
                    protein descriptors for '+pro_id)
        protein_id_error_index.append(idx)
        continue
        result.append(ALL)
#print(pro_id,':',ALL)

 protein_id = pd.Series(protein_id)
 protein_id = protein_id.drop(labels=
                     protein_id_error_index).reset_index()

 df1= pd.concat([protein_id,pd.DataFrame(result)],axis=1)
                               .drop(labels=['index'],axis=1)
 array = df1.values
 X_test = array[:,1:]



#pd.DataFrame({'Protein_id_error':protein_id_error_index})
                     .to_csv('propy error_uniprot_TAS2R.csv')

 import pandas as pd
# # df = pd.read_csv('wine.data.csv', header=None)

 from sklearn.metrics import classification_report,
                          confusion_matrix, accuracy_score
 from sklearn.preprocessing import StandardScaler
#import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import statistics as st

import numpy as np
from sklearn.metrics import jaccard_similarity_score
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest,
    f_classif,SelectFpr,chi2,SelectFdr,SelectFwe
from sklearn.linear_model import LogisticRegression



##dataset = pd.read_csv('gpcr_nongpcr_new.csv', header=None)

df = pd.read_csv('GPCRPenDb_All.csv', header=0)
# ff = feature_Selection(df.iloc[:,1:])
# # #print(list(ff))
# # #print(feature_ranking(ff))
# imp_feat=feature_ranking(ff)[0:990]
# imp_feat = np.array(imp_feat)+1
# imp_feat = np.insert(imp_feat,0,0)
array = df.values
X_train = array[:,1:-1]
```

```
#X_imp=X[:,imp_feat]
#print(X_imp)
#print(X_imp)
Y = array[:,-1]



 sc=StandardScaler()
 trainingArrays=sc.fit_transform(X_train)
#print(trainingArrays)
 X_test =sc.transform(X_test)


#print(X)

# Feature Scaling
 sc = StandardScaler()
#X = sc.fit_transform(X_test)


 from sklearn.svm import SVC

# params={'kernel':'linear','class_weight':'auto'}
# svclassifier = SVC(kernel='linear',class_weight={'Class D':0.5})
 model= SVC(kernel='rbf', decision_function_shape='ovo')
# rfe = RFE(svclassifier, 500)
# svclassifier1 = rfe.fit(trainingArrays , training_labels)
model = model.fit(trainingArrays, Y)
predictions = model.predict(X_test)
df_list =pd.Series(predictions)
#df_list.to_csv('results_adaboost.csv')
#df_list.to_csv('results_logistic_ lbfgs.csv')
print(predictions)
#print(classification_report(test_labels, predicted_class_list))
#df_list =pd.Series(predictions)
#df_list.to_csv('results_pca-isa.csv')
#print(df_list)
for seq in predictions:
        if seq == 'CNT':
        print('It is a non-GPCR but',seq,'\n \n')
        elif seq == 'CN':
        print('It is a non-GPCR but', seq, '\n \n')
        elif seq[0] == 'G':
        f = seq.split('_')
print('It is a {0} protein and belongs to; \n family:
    {1} \n subfamily:  {2} \n subsubfamily: {3}  \n subtype:
    {4} \n \n'.format(f[0],f[1],f[2],f[3],f[4]))

#print(df_list)
```

Appendix O- Relabeling the Class Column for Final training set

```
import pandas as pd
import numpy as np

outF = open("relabelling_all gpcrs_new.csv", "w")
with open('gpcrpendb_results_all gpcrs_edit(errors reomoved)
                                        .csv', mode = 'r') as vcf:
d = []
for line in vcf:
#if line == "ID":
a = line.split(',')
c= a[1]
b = "GPCR" + '_' + a[10] + "_" + a[11] + "_" + a[12] + "_" +
                                                        a[13]
outF.write(b )
```

Appendix P- Screening for Differences in IDs Between Two Csv Files

```
import pandas as pd



df = pd.read_csv('SOCN_list.csv')
mylist = df['ID'].tolist()
mylist2 = df['Protein_id'].tolist()
print(mylist)
print(mylist2)
newlist = []
for idx, id in enumerate(mylist,2):
for num in mylist2:
# print(index)
if idx == num:
print("id =", idx,id)
newlist.append(id)
else:
continue
print(len(newlist))
pd.DataFrame(newlist).to_csv("new_66_difference.csv",
                                        index = False)
print("newlist =", newlist)
new_list = list(set(mylist) - set(newlist))
print(new_list)
print("length of new_list =", len(new_list))



def list_diff(list1, list2):
```

```
out = []
for ele in list1:
if not ele in list2:
out.append(ele)
return out

new_list2 = list_diff(mylist, newlist)
print("length of original full list = ", len(mylist))
print("new_list2=", new_list2)
print("length of new_list2 =", len(new_list2))
pd.DataFrame(new_list2).to_csv("new_difference", index = False)
```

Appendix Q- Classification Tables

Table 6.1: ACC and OACC (in %) of GPCR classification at superfamily level

Classifiers	**TPR**	**TNR**	**FPR(TM)**	**FPR(NTM)**	**OACC**
PCA-ISA	97.6	94.5	9.5	2.3	95.41
Log-Reg	98	97	5.3	1.2	97.24
SVM	98.5	98	3.4	0.7	97.8
MLP-NN	98.5	98.5	2.0	0.3	98.08

Table 6.2: ACC and OACC (in %) of GPCR classification at family level

Family	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
A	100	100	100	99.6
B1	97.6	100	99	99.2
B2	97.4	97.8	90.4	97.8
C	94.8	96.6	84.2	97.8
D	35	42.4	20	77.4
E	83	80	0	92
F	87.2	97.4	93	97.8
T2R	100	100	100	100
OACC	98.6	99.1	97.6	99.1

Table 6.3: ACC and OACC (in %) of GPCR classification of Class A family at subfamily level

Family A	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Alicarboxylic acid	96	100	43	100
Aminergic	100	100	98.2	99.8
Lipid	94.6	98.2	92.2	90.4
Melatonin	100	100	93.4	83.8
Nucleotide	96.8	96.8	93.2	92.0
Orphan	95.2	94.6	92.8	93.0
Peptide	96.8	98	98.8	97.8
Protein	100	99.4	98.4	98.6
Sensory	100	98	95.2	97.6
Steroid	100	99.2	59	100
OACC	97.33	97.85	95.75	96.27

Table 6.4: ACC and OACC (in %) of GPCR classification of Class A subfamily Alicaboxylic at sub-subfamily level

Alicaboxylic	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Hydroxicarboxylic acid	100	100	100	100
Oxoglutarate receptors	100	100	100	100
Succinate receptors	100	100	100	100
OACC	100	100	100	100

Table 6.5: ACC and OACC (in %) of GPCR classification of class A subfamily Aminergic at sub-subfamily level

Aminergic	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
5-Hydroxytryptamine	98	96	95.4	99.2
Acetylcholine	95.8	100	94.2	100
Adrenoceptors	92.8	100	98	98.2
Dopamine	98.2	100	100	96.4
Histamine	93.2	97.2	90	100
Trace amine	100	100	100	90
OACC	95.83	98.61	96.39	98.21

Table 6.6: ACC and OACC (in %) of GPCR classification of class A subfamily Lipid at sub-subfamily level

Lipid	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Cannabinoid	100	100	93.4	100
Free fatty acid	92	100	100	100
GPR18 - GPR55	68.4	76	61.4	100
Leukotriene	75	100	87	85.6
Lysophospholipid (LPA)	95	100	100	100
Lysophospholipid (S1P)	100	93.4	88.6	86.4
Platelet-activating factor	100	100	100	100
Prostanoid	97.5	100	100	100
OACC	92.56	97.67	93.02	96.74

Table 6.7: ACC and OACC (in %) of GPCR classification of class A subfamily Nucleotide at sub-subfamily level

Nucleotide	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Adenosine	100	100	100	100
P2Y	96.4	100	100	100
OACC	97.78	100	100	100

Table 6.8: ACC and OACC (in %) of GPCR classification of class A subfamily Protein at sub-subfamily level

Protein	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Chemerin	100	100	99	100
Chemokine	98	100	100	100
Glycoprotein homorne	100	100	100	100
Prokineticin	100	100	100	100
OACC	99.07	100	99.08	100

Table 6.9: ACC and OACC (in %) of GPCR classification of class B1 subfamily Peptide at sub-subfamily level

Peptide	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Calcitonin receptors	100	100	43	100
Corticotropin-rel. fct.	100	100	100	100
Glucagon receptor	100	100	100	100
Parathyroid hormone	100	100	83	100
VIP and PACAP	100	100	86	100
OACC	100	100	96.36	100

Table 6.10: ACC and OACC (in %) of GPCR classification of family A subfamily Alicaboxylic sub-subfamily 5-hydroxytryptamine at subtype level

5-hydroxytryptamine	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
5-HT$_{1A}$	100	100	100	100
5-HT$_{1B}$	100	96	100	76
5-HT$_{1D}$	100	100	80	100
5-HT$_{1E}$	50	60	60	76
5-HT$_{1F}$	100	66.8	90	88.4
5-HT$_{2A}$	78.4	96.6	95	76
5-HT$_{2B}$	81.8	60	40	40
5-HT$_{2C}$	100	100	100	100
5-HT$_{4}$	100	100	60	100
5-HT$_{5A}$	100	100	80	100
5-HT$_{7}$	100	100	65	100
OACC	94.54	96.51	91.82	96.36

Table 6.11: ACC and OACC (in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Acetylcholine at subtype level

Acetylcholine	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
M$_1$	91.75	93.40	100	100
M$_2$	93.40	95	96.60	86.80
M$_3$	88.40	100	86.80	93.40
M$_4$	100	100	50	100
M$_5$	100	100	100	100
OACC	90.91	94.54	89.09	94.54

Table 6.12: ACC and OACC (in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Adrenoceptors at subtype level

Adrenoceptors	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
α_{1A}	100	100	100	100
α_{1B}	100	100	93.40	100
α_{1D}	100	100	100	100
α_{2A}	80	70	100	90
α_{2B}	100	100	100	73.40
α_{2C}	100	100	60.80	100
β_1	83.40	100	91	85
β_2	100	100	100	85
β_3	100	100	100	100
OACC	96.19	97.14	91.43	93.33

Table 6.13: ACC and OACC (in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Dopamine at subtype level

Dopamine	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
D_1	100	100	100	100
D_2	100	100	100	100
D_3	100	100	100	100
D_4	100	100	100	100
D_5	100	100	100	100
OACC	100	100	100	100

Table 6.14: ACC and OACC (in %) of GPCR classification of family A subfamily Aminergic sub-subfamily Histamine at subtype level

Histamine	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
H_1	100	100	90	83.40
H_2	100	100	100	100
H_3	95	100	85	100
H_4	94	100	75	100
OACC	97.14	100	85.71	94.29

Table 6.15: ACC and OACC (in %) of GPCR classification of Class A family at subfamily level

Peptide	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Ghrelin	96	100	100	100
Proteinase-activated receptors	100	100	98.2	99.8
Melanocortin receptors	97.6	98.2	92.2	90.4
Somatostatin receptors	100	100	99.4	83.8
Neurotensin receptors	96.8	98.8	97.2	98.0
Neuropeptide Y receptors	98.5	100	98.8	99.0
Galanin receptors	99.8	100	98.8	97.8
Cholecystokinin receptors	100	100	98.4	98.6
Formylpeptide receptors	100	100	98.4	100
Complement peptide	100	100	95.2	97.6
Gonadotropin-releasing hormone	98	99.5	100	100
Angiotensin receptors	99	100	100	100
Motilin receptors	99.8	100	100	100
Orexin receptors	99.7	99	98	99
Thyrotropin-releasing hormones	100	100	100	100
Bombesin receptors	100	100	100	100
Neuromedin U receptors	100	100	100	100
Endothelin receptors	97	99	97.8	98
Bradykinin receptors	99.7	100	99	100
Vaso Pressin & oxytocin	98	99.8	100	100
Tachikinin receptors	100	100	100	100
Apelin receptors	100	100	100	100
Opioid receptors	100	99.8	100	100
Neuropeptide W/B receptors	99.8	100	100	100
Urotensin receptors	96	98.3	97.8	98
Prolactin-releasing peptide receptors	100	100	100	100
Peptide P518 receptors	100	100	100	100
Relaxin family peptide receptors	97.5	100	100	100
Neuropeptide S receptors	99.8	99.9	100	100
Melanin-concentrating hormone	97.7	99.8	98	99
Kisspeptin receptors	98	100	98	100
Neuropeptide FF/AF receptors	97	99	96	98
OACC	98.43	99.85	98.75	99.27

Table 6.16: ACC and OACC (in %) of GPCR classification of family C at subfamily level

Class C	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
Amino Acid receptor	100	100	100	100
Ion receptor	100	100	100	100
Orphan receptor	100	100	100	100
Sensory receptor	100	100	100	100
OACC	100	100	100	100

Table 6.17: ACC and OACC classification of family C subfamily Amino acid at sub-subfamily level

Amino Acid	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
$GABA_B$ receptors	100	100	100	100
Metabotropic glutamate receptors	100	100	100	100
OACC	100	100	100	100

Table 6.18: ACC and OACC (in %) of GPCR classification of family C subfamily Amino acid sub-subfamily MGluR at subtype level

MGluR	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
$MGluR_1$	100	100	100	100
$MGluR_2$	100	100	100	100
$MGluR_3$	100	100	100	100
$MGluR_4$	100	100	100	100
$MGluR_5$	100	100	100	100
$MGluR_6$	100	100	100	100
$MGluR_7$	100	100	100	100
$MGluR_8$	100	100	100	100
OACC	100	100	100	100

Table 6.19: ACC and OACC (in %) of GPCR classification of family F subfamily Protein receptor sub-subfamily Frizzled at subtype level

Frizzled	**PCA-ISA**	**Log-Reg**	**SVM**	**MLP-NN**
FZD_1	100	100	100	100
FZD_2	100	100	100	100
FZD_3	100	100	100	100
FZD_4	100	100	100	100
FZD_5	100	100	100	100
FZD_6	100	100	100	100
FZD_7	100	100	100	100
FZD_8	100	100	100	100
FZD_9	100	100	100	100
FZD_10	100	100	100	100
SMO	100	100	100	100
OACC	100	100	100	100

Curriculum Vitae

Fredrick Ayivor was born on December 08, 1988. The first son of Emmanuel Ayivor and Vida Amankwah, he graduated from St Thomas High School at Osu, Ghana, in 2007. He entered Kwame Nkrumah University of Science & Technology (KNUST) in the fall of 2008. After pursuing his bachelor's degree in Mathematics in the summer of 2012, he worked as a Teaching Assistant, at the same institution (KNUST).

In the fall of 2014, he entered the Graduate School of New Mexico State University at las cruces, New Mexico. While pursuing a master's degree in Mathematical Science he worked as a Teaching Assistant, and as the Laboratory Instructor for the institution. Fredrick got accepted in the Computational Science program at University of Texas at El Paso to pursue his Doctoral degree in fall 2016 through which he obtained a Master's degree in 2018. He currently resides in El Paso with his wife and child.

Email address: **fayivor@miners.utep.edu**