2023-08-01

Comparative Study Of Supervised Classification Techniques With A Modified Knn Algorithm

Noah Owusu
University of Texas at El Paso

COMPARATIVE STUDY OF SUPERVISED CLASSIFICATION TECHNIQUES WITH A MODIFIED KNN ALGORITHM

NOAH OWUSU

Master's Program in Mathematical Sciences

APPROVED:

Abhijit Mandal, Ph.D., Chair.

Suneel Babu Chatla, Ph.D.

Sourav Roy, Ph.D.

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

To my

MOTHER, Mercy Oppong

with much love

COMPARATIVE STUDY OF SUPERVISED CLASSIFICATION TECHNIQUES WITH A MODIFIED KNN ALGORITHM

by

NOAH OWUSU

THESIS

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

MASTER OF SCIENCE

Department of Mathematical Sciences

THE UNIVERSITY OF TEXAS AT EL PASO

August 2023

Acknowledgement

I am immensely grateful to the almighty God for overseeing my triumphant journey. I would like to express my profound appreciation to Dr. Abhijit Mandal, my thesis advisor from the Department of Mathematical Sciences at The University of Texas at El Paso, for his invaluable guidance throughout this project. Also, I extend my sincere thanks to Dr. Suneel Babu Chatla of the Department of Mathematical Sciences and Dr. Sourav Roy of the Department of Biological Sciences at The University of Texas at El Paso for their exceptional contributions.

Abstract

The goal of classification is to develop a model that can be used to accurately assign new observations to labeled classes based on the patterns learned from the training data. K-nearest Neighbors algorithm (KNN) is a popular and widely used algorithm for classification, however, its performance can be adversely affected by the presence of outliers in a dataset. In this study we have modified this existing KNN algorithm that can alleviate the effect of outliers in a dataset, thereby improving the performance of the KNN algorithm. We compared the performances of the Modified KNN method and the Existing KNN algorithm as well as other six machine learning algorithms – Naive Bayes algorithm, Random Forest, Support Vector Machine (SVM Linear), Logistic Regression (logit), Linear Discriminant (LDA), and Quadratic Discriminant Analysis (QDA). Utilizing a simulated data and HCV dataset which is available at UCI machine learning repository (*HCV data* 2020), we compared the performances of these techniques in terms of F_1 score, AUC-ROC, and accuracy. The simulation study revealed that the Modified KNN method outperforms the existing KNN algorithm when applied to a simulated datasets that contained different proportion of outliers. Also, with the real data, the Modified KNN method outperforms the existing KNN algorithm in predicting Hepatitis C. The performance evaluations confirm the validity of the Modified KNN method.

Keywords: Classification, K-nearest Neighbors algorithm (KNN), modified KNN method, UCI, HCV, evaluation metrics.

Contents

List of Tables

Chapter 1

Introduction

The process of classifying data into several groups according to a pre-established set of rules is known as statistical classification. In the domains of machine learning, data mining, and other related research, it is a fundamental idea (Hastie et al. 2009). The goal in classification is to take an input vector $\mathbf{X}$ and to assign it to one of K disjoint class C_k, where $k = 1, \cdots, K$. The input space is thereby divided into decision regions whose boundaries are called decision boundaries or decision surfaces (Bishop & Nasrabadi 2006). Using the pattern and relationship discovered from the training data, the objective of statistical classification is to build a model that can precisely predict the class labels of incoming instances. An optimal classification procedure should whenever possible account for the cost associated with the misclassification.

For data set $\mathbf{D}$, which contains the following objects, the categorization process is carried out:

- Set size $\rightarrow A = \{A_1, A_2, \cdots |A|\}$, where $|A|$ is the number of attributes or the size of set A.
- Class label $\rightarrow C = \{c_1, c_2, \cdots |C|\}$, where $|C|$ is the number of classes and $|C| \geq 2$.

Given the data set $\mathbf{D}$ the main of goal machine learning is to construct a classification or prediction function to connect the values of attribute in $\mathbf{A}$ and classes in $\mathbf{C}$.

Statistical classification has been categorized into two: Supervised and Unsupervised. In supervised classification, the model is trained using a set of training instances, where each instance is related to a specific label class. The model uses this information to learn underlying pattern and relationship in the data and then applies these patterns to new, un-

labelled data to make predictions about their class labels. Popular supervised classification algorithms include logistic regression, decision tress, and support vector machine. Unsupervised classification is used when there are no pre-labeled samples and the objective is to find hidden structures and patterns in the data without been aware of the classes beforehand. Unsupervised classification algorithms organize the data into clusters based on similarity, and each cluster represents a candidate class. The two commonly used unsupervised classification technique are K-means clustering and hierarchical clustering. Generally, statistical classification is a powerful tool for identifying patterns and relationship in a data and can be applied to wide range of real-world problems, such as image and speech recognition, spam filtering, fraud detection and more (James et al. 2013).

1.1 Problem Statement and Goals

K-nearest Neighbors algorithm (KNN) is a popular and widely-used algorithm for classification and regression tasks. KNN uses proximity to make classifications or predictions about the grouping of an individual data point. It works off the assumption that similar point can be found near one another. However, its performance can be adversely affected by the presence of outliers in a dataset. Outliers are data points that differ significantly from the majority of the data, and they can have a substantial impact on the KNN algorithm due to its reliance on local averaging. A study conducted by (Jiang & Zhou 2004) pointed out that KNN classifiers are sensitive to outliers and noise contained in training data. Therefore, they proposed approaches that can be used to edit the training data so that the performance of the classifiers can be improved. Also, due to KNN's sensitivity to outliers, a study conducted by (Su & Tsai 2011) stated that distance-based approaches, which includes KNN are popular in outlier detection algorithms in the computer science literature.

In view of this problem of KNN algorithm, the ultimate goal of this study is to modify the existing KNN algorithm that seek to dampen the effect of outliers in datasets, thereby

improving the performance of the algorithm. The modified method which we termed as Modified KNN algorithm will allow us to predict the outcome of the new observation by fitting a generalized linear model (with the Bernoulli distribution) to the training set that correspond to the k-nearest neighbors based on their distances. Specifically, this study aims to:

1 Modify the existing KNN algorithm

2 Compare the modified method with already existing methods for classification using both real and simulated data

Chapter 2

Existing Classification Methods

As this thesis seeks to compare existing classification techniques and our proposed method, we consider seven existing standard classification techniques: Naive Bayes algorithm, K-Nearest Neighbors (KNN), Random Forest, Support Vector Machine (SVM),Logistic Regression (logit), Linear Discriminant Analysis, and Quadratic Discriminant Analysis. For details of these methods, we refer to Hastie et al. (2009). The chosen methods are briefly discuss below:

2.0.1 Naive Bayes Algorithm

Naive Bayes classifier is a popular machine learning algorithm that is used for classification task such as text classification. It belongs to the family of generative learning algorithms, which means that it models the distribution of inputs for a given class or category. This approach works on the principle of conditional probability as given by Bayes theorem and the adjective naive refers to the assumption that the features in a dataset are mutually independent. In reality, the independence assumption is often violated, although naive Bayes classifiers nevertheless perform very well under this unrealistic assumption.

Mathematically, let k be a category of interest out of a set of categories $l = \{1, \cdots, L\}$. Let $X_i = (x_{i1}, x_{i2}, \cdots, x_{ip})^T$ be a vector of p predictors and Y_i be a vector of response variable for individuals $i \in \{1, \cdots, n\}$.

$$P(Y_i = k|X_1, \cdots, X_n) = \frac{P(Y_i = k)P(X_1, \cdots, X_n \mid Y_i = k)}{\sum_l P(Y_i = l)P(X_1, \cdots, X_n|Y_i = l)}. \tag{2.1}$$

The model's naivety allows us to replace the likelihood terms with the product the

probabilities of X_i given Y_i.

$$P(Y_i = k|X_1, \cdots, X_n) = \frac{P(Y_i = k)\prod_i P(X_i|Y_i)}{\sum_l P(Y_i = l)\prod_i P(X_i|Y_i)}. \tag{2.2}$$

To estimate the probability that Y_i belongs to category k given $\{X_1, \cdots, X_n\}$, we can substitute estimates of elements of the right-hand side of the above equation.

The proportion of the training set that falls under category k can be substituted to estimate the prior probability. That is:

$$\hat{P}(Y_i = k) = \frac{1}{n}\sum_i I(Y_i = k), \tag{2.3}$$

where, I is an indicator variable via which the response variable is coded.

There are various methods for estimating the likelihood terms. In this research, our final Naive Bayes model uses Nadaraya-Watson as used by Wigglesworth (2018) to estimate the joint density functions for every unique combination of X_i and Y_i, which we call $f_{j|l(\cdot)}$:

$$\hat{P}(X_i|Y_i = k) = \{\hat{f}_{1|l}(X_{i1}), \cdots, \hat{f}_{p|k}(X_{ip})\}. \tag{2.4}$$

Nadaraya-Watson estimator is a non-parametric method used to estimating a joint density function utilizing a multivariate kernel function such as multivariate Gaussian kernel.

The following assignment rule is used once the estimates are computed. We do not need to take the denominator into account, because only the numerate of the Bayes rule varies across categories.

$$\hat{Y}_i = \underset{k}{\text{argmax}}\left[\hat{P}(Y_i = k)\cdot\prod_i \hat{P}(X_i|Y_i = k)\right]. \tag{2.5}$$

This method is advantageous since they requires small number of training data to estimate parameters, hence they are fast and reliable for making real-time predictions. They can also handle continuous and discrete data. However, they encounter "zero frequency problem". Thus, if a categorical variable has a category in the test dataset that wasn't

included in the training dataset, the model will assign it a 0 probability and will be unable to make a prediction. Also, the assumption that the predictors are conditionally independent is exceedingly implausible in real-world settings. These models are computationally expensive when used to classify large number of items.

2.0.2 Support Vector Machine (SVM)

SVM is a powerful and flexible classification algorithm that seeks to find the best line or decision boundary that can divide n-dimensional space into classes so that we may simply place new data points in the correct category in the future. This optimal decision boundary is called *hyperplane*: one that maximizes the total of the orthogonal distances, or margin, the hyperplane and closest training observation in each category. In some cases, it's not viable to find a hyperplane that completely separates the training observations in p-dimensional space. In these instances, we can opt to apply a kernel, such as a linear, polynomial, or radial bias function kernel, to map the training data to higher dimensions where an acceptable hyperplane can be drawn. This approach helps us to achieve a suitable separation of the training observations (Wigglesworth 2018). The general mathematical definition of hyperplane of a p-dimensional space is given by the equation:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0. \tag{2.6}$$

Thus if a point $X = (X_1, X_2, \cdots, X_p)^T$ in a p-dimensional space (i.e. a vector of length p) satisfies (2.6), the X lies on the hyperplane.

Suppose X does not satisfy (2.6) ; rather,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0, \tag{2.7}$$

then this tells us that X lies on one side of the hyperplane. On the contrary, if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0, \tag{2.8}$$

then X lies on the other side of the hyperplane. Hence, a hyperplane can be seen as a line dividing p-dimensional space into two halves. One can quickly determine on which side of the hyperplane a point lies by computing the sign of the left side of (2.6).

The Maximal Margin Classifier

Typically, there exist an infinite number of hyperplanes that can perfectly separates our data. To create a classifier that uses a separating hyperplane, a reasonable approach is to choose the hyperplane with the maximal margin (i.e. *the optimal margin hyperplane*) from the infinite potential hyperplanes. This hyperplane separates the training observations at the farthest distance. In other words, we measure the distance between each training observation and a given separating hyperplane and select the smallest distance as the *margin*, which is the minimum distance from observations to the hyperplane. The maximal margin hyperplane is the one with the largest margin, which means it has the farthest minimum distance to the training observations. Then classifying a test observation based on which side of the maximal margin hyperplane it lies in know as maximal margin classifier. The maximal margin hyperplane is the solution to the optimization problem below:

$$\underset{\beta_0, \beta_1, \cdots, \beta_p}{\text{Maximize}} \quad M, \tag{2.9}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j = 1, \tag{2.10}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) \geq M \qquad \forall \qquad i = 1, \cdots, n. \tag{2.11}$$

The constraint (2.11) guarantees that each observation will be on the correct side of the hyperplane, provided M is positive. Again, (2.10) is not really a constraint on the hyperplane, since if $\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} = 0$ defines a hyperplane, then so does $k(\beta_0+\beta_1 X_{i1}+\beta_2 X_{i2}+\cdots+\beta_p X_{ip}) = 0$ for any $k \neq 0$. However, (2.10) gives a further context to (2.11). One can demonstrate that with this constraint, the the perpendicular distance from the i^{th} observation to the hyperplane is given by $\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}$.

As a result , the constraints (2.10) and (2.11) guarantee that each observation is on the correct side of the hyperplane and is at least M distance away from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \cdots, \beta_p$ to maximize M.

Support Vector Classifier

The support vector classifier is a method that uses a hyperplane to classify a test observation based on its position relative to the hyperplane. The hyperplane is selected in such a way that it can accurately separate the majority of training observation into two classes, although there may be some misclassified observations. It is the solution to the optimization problem

$$\underset{\beta_0, \beta_1, \cdots, \beta_p, \epsilon_1, \cdots, \epsilon_n,}{\text{Maximize}} \quad M, \tag{2.12}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j = 1, \tag{2.13}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) \geq M(1 - \epsilon_i), \tag{2.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{2.15}$$

where C is a non-negative tuning parameter. It establishes the upper bound of the sum of $\epsilon_i^{'}s$ which determines the number and severity of violations to the margin (and to the hyperplane) that we will tolerate. Similar to (2.11), M is the width of the margin; our goal

is to make this quantity as large as possible. In (2.14), $\epsilon_1, \cdots, \epsilon_n$ are *slack variables* which directly measures the distance from a wrongly classified observations x_i to its corresponding margin. Thus, the slack variables allow individual observations to be on the incorrect side of the margin or hyperplane. After solving the optimization problem, we classify a test observation x^* by simply finding which side of the hyperplane it lies. That is, we classify the test observation according to the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$.

How SVM classifier works

Here are the steps involved in how the SVM classifier works:

1 The SVM classifier starts with input data that consists of a set of labeled training instances. Each instance is represented by a set of features or attributes.

2 The input data may contain a large number of features, which may not be relevant for classification. Therefore, feature extraction is performed to select the most relevant features that can best differentiate between the classes.

3 The selected features are then mapped to high-dimensional space, where the data points can be linearly separable. In a case where the data points may not be linearly separable in the feature space, the SVM uses a kernel function to map the data points to a higher-dimensional space where they can be linearly separable. The most commonly used kernel functions are linear, polynomial, and radial basis function (RBF).

4 The SVM then finds the optimal hyperplane or decision boundary that best separates the data points of different classes. The optimal hyperplane is the one that maximizes the margin between the hyperplane and the nearest data points of each class.

5 Margin: The margin can be hard or soft. Hard margin is that kind of decision boundary that ensures that all data points are classified correctly. While this leads to SVM classifier not causing error, it can cause the margin to shrink thus making

the whole running of the SVM classifier ineffective. Soft margin is a modified version of the margin used in SVM that allows some misclassifications or errors to occur in the training data. The soft margin is implemented by introducing a slack variable or penalty term into SVM optimization problem. The slack variable allows the data points to be classified on the wrong side of the hyperplane, but penalizes the misclassifications by a cost proportional to the distance of the misclassified point from the margin.

The amount of slack variable allowed is controlled by a parameter called regularization parameter C. A smaller value of C allows more misclassifications and leads to wider margin, while a large value of C results in a narrower margin and fewer misclassifications.

2.0.3 Random Forest

The concept of random-subspace was first proposed by Ho (1995) and later evolved into random forest algorithm, which was formally introduced by Breiman (2001). Random forest is a classifier that uses numerous decision trees on different subsets of a given dataset and takes the average to enhance the predicted accuracy of that dataset and it reduces overfitting. It is built on the notion of ensemble learning, which is the process of merging numerous classifiers to solve a complex problem and improve the model's performance. Since this algorithm creates trees, it tells us that the more the trees the more robust forest.

Random forest is one of the best-performing learning algorithms. Random forest tend to predict better than other methods that make assumption of linearity as it easily adapt to nonlinearity detected in the data. More precisely, ensemble algorithms like random forests are well suited for medium to large dataset. The methods for linear regression and logistic regression will not operate when the number of independent variables is greater than the number of observations because there are too many estimated parameters. However, random forest works because not all predictor variables are used simultaneously.

According to (Breiman 2001), random forest algorithm provides a more accurate estimation of the error rate compared to decision trees. In particular, it has been proven mathematically that the error rate consistently converges as the number of trees in the model increases.

When training the random forest algorithm, the error rate is estimated using the out-of-bag (OOB) error. This involves randomly leaving out around one-third of the observations for each tree's bootstrap sample, which creates the OOB sample. By minimizing the OOB error, the algorithm selects the most appropriate parameters for the model, such as the value of m that determines the subset of predictor variables. Tuning these parameters during model selection is crucial to control the final depth of the trees and ensure accurate predictions.

The variable importance of each variable is calculated to gain more insight on the complex model. This involves calculating the total improvement in the objective function for each predictor variable. The improvement is determined based on the splitting criterion used at each internal node of the tree, and it is summed up across all internal nodes in each tree and across all trees in the random forest.

Although, random forest algorithm provides high level of accuracy in predicting outcome, it consumes more time compared to decision tree algorithm and also requires more computational resources.

How Random Forest algorithm works

The algorithm uses the following steps in its execution;

1. Draw multiple random samples (bootstrap samples) with replacement from the original dataset.

2. For each bootstrap sample, creation of decision trees are performed. During this stage, at each node of the tree;

   – A subset of features are randomly selected.

- Find the best split based on these features using an impurity measure (e.g., Gini impurity or entropy).
- The process continues recursively until a stopping criterion is met (e.g., max depth, minimum samples per leaf).

3. Predictions are made by calculating the prediction for each decision tree, then taking the most popular results.

2.0.4 Logistic Regression

Logistic regression is a statistical method that aim to model the posterior probabilities of multiple (K) classes via linear function in the predictor variable (x), by ensuring that the probabilities add up to one and stay within the (0,1) range (Hastie et al. 2009). This method addresses the same question that Discriminant function analysis and Multiple regression do but with no distributional assumptions on the predictors (predictors do not have to be normally distributed, have equal variance in each group). The predictor variables can be continuous or categorical. The goal of logistic regression is to find the best fitting model that can predict the probability of outcome variable based on the predictor variables.

The logistic regression model uses a mathematical function called sigmoid function to estimate the probability of the outcome variable. The sigmoid function maps any real-valued number to a value between 0 and 1, which can be interpreted as the probability of a particular outcome. The logistic regression model estimates the coefficients of the predictor variables to maximize the likelihood of the observed data.

How Binary logistic regression works

A special case of logistic regression is binary logistic regression which seeks to develop a classifier that is capable of making binary decision about new input observation.

Consider a single input observation x, represented by a vector of features $[x_1, x_2, \cdots, x_p]$ and the classifier output y which can be 1 (meaning the observation is a member of the class)

or 0 (the observation is not a member of the class). Our aim is to know the probability $P(y = 1|x)$ that this observation is a member of the class. So maybe the decision is "positive diagnosis" versus "negative diagnosis" for diabetes, the features represent the factors related to diabetes, $P(y = 1|x)$ is the probability that diabetes diagnosis is positive given, and $P(y = 0|x)$ is the probability that diabetes diagnosis is negative.

The way in which logistic regression works is by using a training set to learn a **bias term** and a vector of **weights**. Each weight w_i corresponds to one of the input features x_i, and it has a real number value. The weight represents the significance of that feature in the classification decision, and can be either positive (indicating that the instance belongs to positive class) or negative (indicating that the instance belongs to negative class). The **bias term**, also called **intercept**, is another real number that is added to the weighted inputs (Keselj 2009). Once the weights are learned in the training set, the method first multiplies each x_i by its weights w_i, sum up the weighted features and add up the bias term b. The results denote by z expresses the weighted sum of the evidence of the class of the test instance.

$$z = \left(\sum_{i=1}^{n} w_i x_i + b \right) = \mathbf{w} \cdot \mathbf{x} + b. \tag{2.16}$$

The z is evaluated at the **sigmoid** function to create probability. The sigmoid function is an S-shaped function which is also called **logistic function**. It tends to squash outliers to 0 or 1 since it is virtually linear around 0 and flattens toward the ends. The sigmoid has the following equation,

$$\sigma(z) = \frac{1}{1 + \exp(z)}. \tag{2.17}$$

By ensuring that the two cases, $p(y = 1)$ and $p(y = 0)$, sum to 1, we find probability from (2.17);

$$p(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \exp(\mathbf{w} \cdot \mathbf{x} + b)}, \tag{2.18}$$

$$p(y = 0) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{\exp(-(\mathbf{w} \cdot \mathbf{x} + b))}{1 + \exp(-(\mathbf{w} \cdot \mathbf{x} + b))}. \tag{2.19}$$

The algorithm uses maximum likelihood estimation to estimate the parameters. However, maximizing the log-likelihood of the binary logistic function does not have a closed-form solution. Therefore, we approximately solve it numerically using methods like gradient descent.

2.0.5 Linear Discriminant Analysis (LDA)

LDA is a linear classification method which assumes that the joint density of all features, conditional on the target's class (qualitative response), is a multivariate Gaussian. This means that the density P of the features X, given the target y is in class k, are assumed to be given by

$$P(X|y=k) = \frac{1}{(2\pi)^{p/2} \mid \Sigma \mid^{1/2}} \exp\Big(-\frac{1}{2}(X-\mu_k)^t \Sigma^{-1}(X-\mu_k)\Big). \tag{2.20}$$

where p is the number of features, μ is the mean vector, and Σ_k is the covariance matrix of the Gaussian density for class k. The decision boundary between two classes, say k and l, is the hyperplane on which the probability of the target belonging to either class is the same. This implies that, on this hyperplane, the difference between the two densities (and hence the log-odds ratio between them) should be zero.

An important assumption in LDA is that the Gaussian for different classes share the same covariance matrix $\mathbf{\Sigma}_k = \mathbf{\Sigma} \quad \forall k$. This assumption comes in handy for log-odds ratio calculation as it makes the normalization factors and some quadratic parts in the exponent cancel out. This yields a decision boundary between k and l that is linear in X:

$$\begin{aligned} log\left(\frac{P(y=k|X)}{P(y=l|X)}\right) =& \log\frac{f_k(x)}{f_l(x)} + \log\frac{\pi_k}{\pi_l}, \\ =& \log\frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k+\mu_l)^T\Sigma^{-1}(\mu_k-\mu_l) + x^T\Sigma^{-1}(\mu_k-\mu_l). \end{aligned} \tag{2.21}$$

In order to calculate the density of the features $P(X|y=k)$, we need to first estimate the Gaussian parameters: μ_k as the sample mean ($\hat{\mu}_k$) and covariance matrix Σ as the

empirical covariance matrix $\hat{\Sigma}$. These parameters are estimated based on the training data and we can obtain the probability of the target belonging to class k from the Bayes rule as:

$$P(y = k|X) = \frac{P(X|y = k)P(y = k)}{P(X)}, \tag{2.22}$$

where $P(y = k)$ is the prior probability of the target belonging to class k and can be estimated by the proportion of $k-$class observations in the sample. This LDA method has a **closed-form** solution and therefore has no hyperparameters. From (2.21), we see that the decision rule, with $G(x) = \underset{k}{\text{argmax}}\ \delta_k(x)$ is similar to *linear discriminant functions*:

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k. \tag{2.23}$$

2.0.6 Quadratic Discriminant Analysis (QDA)

In LDA, it is assumed that Gaussian for different classes share the same covariance matrix, which is convenient but may not be correct for certain data. To allow for more flexibility and accuracy, some may prefer to use k covariance matrices to be estimated instead of the common covariance assumption. However, if there are many features, this can greatly increase the number of parameters in the model, which is important to consider since the decision boundaries are functions of these parameters. Additionally, when using multiple covariance matrices, the quadratic terms in the Gaussians' exponent no longer cancel out, resulting in quadratic decision boundaries in X and increased model flexibility. We then get *quadratic discriminant function*(QDA),

$$\delta_k(x) = -\frac{1}{2}log|\mathbf{\Sigma}_k|^{-1} - \frac{1}{2}(x - \mu_k)^T\mathbf{\Sigma}_k^{-1}(x - \mu_k) + \log\pi_k. \tag{2.24}$$

The quadratic equation $\{x : \delta_k(x) = \delta_l(x)\}$ describes the decision boundary between two classes k and l.

2.0.7 KNN Algorithm

The k-nearest neighbors algorithm is a non-parametric, supervised learning classifier that employs proximity to classify or predict the grouping of a single data point. Whiles it can be used to solve either regression or classification problems, it is more widely utilized as a classification technique, working off the assumption that similar points can be found near one another. The KNN algorithm is regarded as a lazy algorithm since it lacks an explicit training stage and defers all computations until prediction (Aha 1997). As a result, the processing of the training examples is postponed until prediction, and training consist essentially of simply storing the training data.

The KNN algorithms compute the class label (classification) or continuous target (regression) based on the k nearest (most "similar") points when making a prediction. Instead of approximating the target function $f(x) = y$ globally, during each prediction, KNN approximates the target function locally. KNN is usually considered as a discriminative model since it does not explicitly try to model the data generating process but models the posterior probabilities ($P(X|f(X))$) directly.

The simplest iteration of the KNN model in the classification context is to forcast the target class as the class label that is most frequently represented among the k most similar training examples for a particular query point. In other words, the class label can be thought of as the "mode" of the k training labels, or as a result of "plurality voting". This is always a majority or a tie in the case of binary predictions (classification problem with two classes). As a result, a majority vote is always a plurality vote.

This technique is useful for it speedy computation and simplicity of execution. It determines its nearest neighbors for continuous data using Euclidean distance, which is defined as:

$$d(\mathbf{X}^{[a]}, \mathbf{X}^{[b]}) = \sqrt{\sum_{j=1}^{m} \left(\mathbf{X}_j^{[a]}, \mathbf{X}_j^{[b]} \right)^2}. \tag{2.25}$$

Despite the classifier's simplicity, the value of 'k' is crucial in classifying unlabeled input. The value of 'K' can be decided through many way, but we just can run the classifier

several times with various value to see which one produces the best results. Because all the calculations are done when the training data is being classified rather than when it appears in the dataset, the computation cost is a little higher.

Upon KNN's easy implementation, it suffer the problem of "Curse of dimensionality". Because of this problem, KNN is also prone to overfitting. While feature selection and dimensionality reduction techniques are used to prevent the problem, the value of 'k' can also have an effect on the model's behavior. Lower k values may overfit the data, but higher k values may "smooth out" the forcast values by averaging the values over a larger area, or neighborhood. However, if the value of 'k' is too large, the data may be underfit.

How KNN algorithm works

The KNN algorithm uses the following steps in its process:

1 The algorithm starts with a labeled dataset containing a set of input features and corresponding output labels. This dataset is used to train the KNN algorithm.

2 Determine the number of nearest neighbors (k) to consider when making predictions. The ideal k value can be determine by cross-validation. It estimates the validation error rate by holding out a subset of training set from the model building process.

3 Compute the distance between the new input sample and all the existing data points in the dataset. The most commonly used distance metric is Euclidean distance, but other metrics like manhattan distance or Minkowski distance can also be used.

4 Select k data points in the dataset that are closest to the new input sample based on the computed distances.

5 The classification is done by determining the majority class among the k-nearest neighbors and assign the new input sample. For regression, we calculate the average of the output values of the k-nearest neighbors and assign that as the predicted value of the new intput sample. The algorithm finally returns the predicted class.

Chapter 3

Modified Method

3.1 Modified KNN Algorithm

The selection of the hyperparameter k is among several factors that impact the effectiveness of the KNN algorithm. When k is too small, the algorithm becomes more sensitive to outliers, whereas a large value for k may result in the inclusion of numerous points from different classes within the neighborhood.

Additionally, determining how to combine the class labels poses another challenge. The most straightforward technique involves using majority voting. However, this can lead to issues when the nearest neighbors differ significantly in their distances, and closest neighbors offer more reliable indications of the object's class.

In view of these issues we have modified the prediction stage of the KNN algorithm process. Modified KNN algorithm will allow us to predict the outcome of the new observation by fitting a generalized linear model (with the Bernoulli distribution) to the training set that correspond to the k-nearest neighbors based on their distance. The distances aid in selecting the set of data required to fit the model. A large k value was chosen to ensure an appropriate linear model fit.

Generalized linear models (GLMs) are particularly useful when the relationship between predictors and the response variable is not linear or when the response variable follow a non-normal distribution. In GLMs, the response variable is assumed to follow a distribution from the exponential family, such as the Gaussian, binomial, Poisson, or gamma distribution (Myers & Montgomery 1997). The key components of a GLM are the linear predictor, the link function, and the variance function. The linear predictor represents the relationship

between the predictors and the response variable, similar to linear regression. The link function maps the linear predictor to the mean of the response variable, enabling the model to accommodate different types of the relationships. The *logit* link function is utilized in this study. The variance function specifies the relationship between the mean and the variance of the response variable. GLMs are estimated using maximum likelihood estimation, and varoius techniques, such as iterated reweighted least squares, are employed for parameter estimation.

Generally, GLMs provide a flexible and powerful framework for modeling data with different types of response variables, allowing for more accurate and interpretable predictions in a wide range of applications, including regression and classification analysis.

3.2 Model Evaluation Metrics

In order to assess the performance of the machine learning models and generalize their capabilities, we considered three machine learning evaluation metrics: F_1 score, Area under the ROC curve (AUC), Accuracy.

3.2.1 F_1 Score

F1 score is a commonly used performance metric in classification task, such as evaluating the performance of machine learning models. A study conducted by Whalen et al. (2016) made use of F_1 score as an evaluation measure when predicting enhancer-promoters interactions in genomics. F_1 score is calculated as the harmonic mean of precision and recall. It provides a single value that considers both precision and recall, giving equal importance to both measures. Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It focuses on accuracy of positive predictions. Recall, also known as sensitivity, measures the proportion of true positive predictions out of all positive samples in the dataset. It focuses on model's ability to identify positive samples. Mathematically, F_1 Score is calculated as below:

$$F_1 \text{ score} = 2 \times (Precision \times Recall)/(Precision + recall).$$

3.2.2 Area under ROC curve (AUC)

Area under the receiver operating characteristic (ROC) curve (AUC) is a performance metric for classification problems at different threshold settings. ROC is a probability curve that helps us to visualise the performance of a classification model and AUC represent the degree of separability. It tells how much the model is capable of distinguishing between classes. AUC is defined as a function of True Positive Rate (TPR), that is, the fraction of true labels that are correctly classified as true and the False Positive Rate (FPR), that is, the probability that a classifier is incorrect when it labels an instance as true. AUC ranges from 0 to 1, where 0.5 indicates a model with no discrimination ability, and 1 indicates a model with perfect discrimination. A higher AUC indicates a better-performing model with strong ability to classify positive and negative samples correctly. AUC-ROC is insensitive to class imbalance. It provides an aggregated performance measure that is affected by the distribution of positive and negative samples.

3.2.3 Accuracy

The accuracy is a common evaluation measure used for classification models. It assesses the overall correctness of the model's prediction by calculating the proportion of correctly classified instances over the total number of instances in the dataset. Mathematically, accuracy is calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{3.1}$$

The accuracy metric provides a simple and intuitive measure of how well the classification model performs in terms of correctly predicting the class labels. A higher accuracy indicates a more accurate and reliable model, while a lower accuracy suggests that the model's accuracy is less reliable. However, accuracy metric may not always provide a

completed picture of a classification model's performance, especially when dealing with imbalanced datasets.

Chapter 4

Simulation Study

In this section we run a simulation experiment to verify the performance of the models based on F_1 score, AUC-ROC, and accuracy measures. We considered three different set of explanatory variables in this particular study, so $p = 2,\ \ 5,\ \$ and $\ 10$, where p is the number of explanatory variables in each set. The explanatory variables are distributed according to standard normal distribution $\mathcal{N}(0, 1)$. The response variable Y_i was generated from a Bernoulli distribution $Bernoulli(\theta)$, where p is given as:

$$\theta = \frac{\exp(\boldsymbol{X} \cdot \beta_0)}{1 + \exp(\boldsymbol{X} \cdot \beta_0)}, \tag{4.1}$$

where, $\boldsymbol{X}$ is $n \times p + 1$ design matrix whose columns contains the values of explanatory variables. β_0 is vector of the true values of the parameters and it is randomly generated from a standard normal distribution with size $p + 1$, where p is the number of explanatory variables. Thus $\beta_0 \sim \mathcal{N}_{p+1}(0, 1)$. A sample size of $n = 100,\ \$ and $\ 200$ was considered.

We checked the performance of the estimators both in pure and contaminated data. We inserted ϵ proportion of outliers to the response variable Y for the contaminated data. To ensure our data is contaminated, we flipped the response variable Y values of 0 to 1 and 1 to 0. We considered the values of the contamination proportion as $\epsilon_i = 0,\ \ 0.05,\ \$ and $\ 0.1$.

The dataset was partitioned into training and test data with a ratio of approxomately $2 : 1$ on the sample. Thus, approximately 67% of the dataset were sampled without replacement for the training process whiles the remaining 33% were used for the testing process.

We have applied the generated data with different set of predictors and different thresh-

old of outlier to KNN Algorithm, Naive Bayes classifier, Random Forest algorithm, SVM Linear, Logistic Regression, LDA, QDA, and Modified KNN. The simulations run 100 times each starting with a random set of train and test data and each model's F_1 score, AUC-ROC, and accuracy are computed. Hence, 100 classification percentage values are achieved. To evaluate the model's classification performance, the average values of the 100 runs are taken with respect to each evaluation metric considered. The R code used for this simulation can be found in the Appendix.

4.1 Simulated Results

In this section, we have reported the model's F_1 score, AUC-ROC, and accuracy for the testing data using sample size of $n = 50$ and $n = 200$ for different number of features (i.e., $p = 2, \quad 5, \quad \text{and} \quad 10$) and different proportion of outliers (i.e., $\epsilon_i = 0, \quad 0.05, \quad \text{and} \quad 0.1$) in the simulated data. Our main idea is to check the performance of the models when the proportion of outliers is changed.

Table 4.1 shows the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models considered and the Modified KNN method applied to the simulated data with $n = 100, \quad p = 2$ and different proportion of outliers (i.e., $\epsilon_i = 0, \quad 0.05, \quad \text{and} \quad 0.1$). The Modified KNN seems to perform similarly as the existing KNN and other models across all the performance metrics when the simulated data contains no outlier ($\epsilon = 0$). With 5% of outliers contained in the simulated data, the Modified KNN is the clear winner in term of correctly predicting the class labels across all the three performance metrics. Once again, when the simulated data contained 10% outlying values, and across all the performance metrics considered the Modified KNN performed best.

Table 4.2 illustrates the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models considered and the Modified KNN method applied on the simulated data with $n = 200, \quad p = 2$ and different proportion of

outliers (i.e., $\epsilon_i = 0,\ \ 0.05,\ \ \text{and}\ \ 0.1$). The results on Table 4.2 favors the Modified KNN in terms of best performance as Table 4.1 across all the performance metrics and when the simulated data contained 5% and 10% outlying values. However, when there is 0% outlier in the simulated data, existing methods like Naive bayes, SVM Linear, LDA and QDA performed wwell.

Table 4.1: Summary of model performance in simulated data with $n = 100, p = 2$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.83	0.78	0.79	0.82	0.76	0.78	0.81	0.79	0.79
`Naive Bayes`	0.84	0.78	0.80	0.84	0.78	0.80	0.82	0.79	0.80
`Random Forest`	0.81	0.76	0.77	0.82	0.78	0.79	0.80	0.79	0.79
`Logistic Regression`	0.84	0.78	0.80	0.75	0.65	0.68	0.67	0.62	0.63
`SVM Linear`	0.84	0.78	0.80	0.76	0.67	0.70	0.67	0.62	0.63
`LDA`	0.84	0.79	0.80	0.74	0.65	0.68	0.67	0.63	0.63
`QDA`	0.84	0.79	0.80	0.81	0.73	0.76	0.79	0.74	0.75
`Modified KNN`	0.84	0.79	0.80	0.84	0.81	0.82	0.84	0.83	0.83

Table 4.2: Summary of model's performance of simulated data with $n = 200, p = 2$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.83	0.77	0.79	0.83	0.79	0.80	0.82	0.80	0.81
`Naive Bayes`	0.85	0.80	0.81	0.84	0.80	0.81	0.83	0.79	0.80
`Random Forest`	0.82	0.76	0.77	0.81	0.78	0.79	0.81	0.79	0.79
`Logistic Regression`	0.85	0.79	0.81	0.73	0.64	0.66	0.69	0.64	0.64
`SVM Linear`	0.85	0.80	0.81	0.74	0.66	0.68	0.69	0.63	0.64
`LDA`	0.85	0.80	0.82	0.71	0.63	0.65	0.68	0.64	0.64
`QDA`	0.85	0.80	0.82	0.81	0.74	0.76	0.80	0.74	0.75
`Modified KNN`	0.84	0.79	0.80	0.84	0.81	0.82	0.84	0.82	0.82

Table 4.3 depicts the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models and the Modified KNN method applied on the simulated data with $n = 100$, $p = 5$ and different proportion of outliers (i.e., $\epsilon_i = 0,\ 0.05,$ and 0.1). In the F_1 score, AUC-ROC, and accuracy perspective and across all proportion of outliers considered for the simulated data, Modified KNN outperforms the existing KNN even though some existing models like random forest and naive bayes similarly performed as the Modified KNN..

Table 4.4 illustrates the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models considered and the Modified KNN method applied on the simulated data with $n = 200$, $p = 5$ and different proportion of outliers (i.e., $\epsilon_i = 0,\ 0.05,$ and 0.1). Table 4.4 produce similar results as Table 4.3 even with $n = 200$. In particular, the Modified KNN outperforms the existing KNN across all the performance metrics and across all the proportion of outliers considered.

Table 4.3: Summary of model's performance of simulated data with $n = 100, p = 5$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.80	0.73	0.75	0.77	0.72	0.73	0.77	0.75	0.75
`Naive Bayes`	0.82	0.76	0.78	0.79	0.71	0.74	0.79	0.71	0.74
`Random Forest`	0.82	0.76	0.78	0.80	0.77	0.77	0.80	0.79	0.79
`Logistic Regression`	0.83	0.78	0.80	0.70	0.64	0.65	0.70	0.67	0.68
`SVM Linear`	0.83	0.78	0.80	0.70	0.64	0.65	0.70	0.67	0.68
`LDA`	0.83	0.79	0.80	0.69	0.63	0.65	0.70	0.67	0.68
`QDA`	0.81	0.76	0.77	0.76	0.69	0.71	0.77	0.70	0.72
`Modified KNN`	0.82	0.77	0.78	0.78	0.76	0.76	0.80	0.78	0.79

Table 4.4: Summary of model's performance of simulated data with $n = 200, p = 5$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.82	0.75	0.77	0.81	0.76	0.77	0.81	0.78	0.79
`Naive Bayes`	0.83	0.78	0.80	0.82	0.74	0.77	0.80	0.73	0.75
`Random Forest`	0.82	0.77	0.78	0.82	0.78	0.79	0.82	0.80	0.81
`Logistic Regression`	0.85	0.80	0.82	0.72	0.64	0.66	0.72	0.68	0.68
`SVM Linear`	0.85	0.80	0.82	0.73	0.65	0.67	0.72	0.68	0.69
`LDA`	0.85	0.80	0.82	0.72	0.64	0.66	0.72	0.68	0.69
`QDA`	0.84	0.79	0.81	0.79	0.70	0.73	0.79	0.72	0.74
`Modified KNN`	0.82	0.78	0.79	0.82	0.79	0.80	0.82	0.80	0.81

Table 4.5 shows the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models considered and the Modified KNN method applied on the simulated data with $n = 100$, $p = 10$ and different proportion of outliers (i.e., $\epsilon_i = 0,\ \ 0.05,\ \ \text{and}\ \ 0.1$). Once again, the Modified KNN performed better than the existing KNN given the performance metrics across the different set of data. Other existing models like SVM linear, LDA also performed well even better than the Modified KNN.

Table 4.6 illustrates the correct classification proportion values in terms F_1 score, AUC-ROC, and accuracy of the existing classification models considered and the Modified KNN method applied on the simulated data with $n = 200$, $p = 10$ and different proportion of outliers (i.e., $\epsilon_i = 0,\ \ 0.05,\ \ \text{and}\ \ 0.1$). The uniqueness about Table 4.6 is that there is no clear winner between the Modified KNN and the existing KNN given the performance metrics and even across the proportion of outliers considered in the simulated data. However, some existing models like SVM linear and LDA clearly outperforms the Modified KNN when there is 0% outlier in the data.

Table 4.5: Summary of model's performance of simulated data with $n = 100, p = 10$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.64	0.71	0.74	0.61	0.69	0.72	0.68	0.73	0.74
`Naive Bayes`	0.66	0.73	0.75	0.63	0.66	0.70	0.51	0.66	0.67
`Random Forest`	0.64	0.72	0.74	0.66	0.73	0.74	0.71	0.74	0.74
`Logistic Regression`	0.75	0.80	0.82	0.66	0.72	0.73	0.71	0.74	0.74
`SVM Linear`	0.75	0.80	0.81	0.68	0.74	0.75	0.72	0.75	0.75
`LDA`	0.76	0.81	0.82	0.68	0.73	0.74	0.72	0.75	0.75
`QDA`	0.63	0.71	0.74	0.59	0.69	0.71	0.62	0.70	0.70
`Modified KNN`	0.67	0.74	0.75	0.71	0.75	0.76	0.74	0.76	0.76

Table 4.6: Summary of model's performance of simulated data with $n = 200, p = 10$ and different values of ϵ.

Method	$\epsilon = 0$			$\epsilon = 0.05$			$\epsilon = 0.1$		
	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**	F_1 **score**	**AUC**	**ACC**
`KNN`	0.66	0.74	0.76	0.69	0.74	0.75	0.74	0.77	0.78
`Naive Bayes`	0.72	0.78	0.80	0.55	0.69	0.72	0.49	0.66	0.69
`Random Forest`	0.69	0.75	0.78	0.72	0.76	0.77	0.74	0.78	0.78
`Logistic Regression`	0.78	0.82	0.83	0.71	0.75	0.76	0.74	0.77	0.77
`SVM Linear`	0.78	0.82	0.83	0.72	0.76	0.76	0.75	0.77	0.78
`LDA`	0.79	0.83	0.84	0.72	0.76	0.76	0.75	0.77	0.77
`QDA`	0.72	0.78	0.79	0.66	0.73	0.75	0.66	0.73	0.75
`Modified KNN`	0.68	0.74	0.74	0.72	0.75	0.76	0.75	0.77	0.77

Chapter 5

Real Data Analysis

In this section we explore the performance of the existing classification mehtods as well as the modified KNN method by applying them on a real dataset. We compare the performance of these methods in term of F_1 score, AUC-ROC, and accuracy measures.

5.1 Data Source and Description

We considered the HCV dataset which is available at UCI machine learning repository (*HCV data* 2020). The dataset contain 615 instances and 14 attributes which summarize laboratory values of blood donors and Hepatitis C patients and demographic values like age. However, the first column of the dataset which is the patient id number was excluded because it is not important in predicting the target variable. The target attribute for classification is `Category`: Blood donor (including blood donor and suspect blood donor) versus Hepatitis C (including its progress "just" Hepatitis C, Fibrosis, Cirrhosis). We coded blood donor (no Hepatitis C) as 0 and Hepatitis C as 1. Our aim was to make medical diagnosis of Hepatitis C based on the results from of the lab work.

In order to evaluate the dataset we did some preprocessing operation, the reason to preprocessing was to summarize the data in the best and suitable way for our algorithms. Missing values of some of the features were adjusted using predictive mean matching imputation method through the *mice* package in R software. Our early exploration of the data revealed that the variable sex does not determine whether a patient is a healthy blood donor or Hepatitis C (p value $= 0.098$ *at* $\alpha = 0.05$). Therefore, the variable sex was discarded during the analysis. Also, it was evident that the features have different range and

variations, so scaling was performed on the features to ensure standard unit of the features. Table 5.1 summarizes the variables included in the dataset dataset post-cleaning. The abbreviated variables refer to laboratory data. Table 5.2 describes the tuning parameters for each model considered in this study.

Table 5.1: Attributes of HCV dataset.

Variable	Description
`Category(diagnosis)`	Binary variable indicating a patient is diagnose of Hepatitis C or not
`Age`	Continuous variable indicating age of the patient
`Sex`	Categorical variable with two levels indicating gender of the patient
`ALB`	Albumin Blood Test
`ALP`	Alkaline Phosphatase
`ALT`	Alanine Transaminase
`AST`	Aspartate Transaminase
`BIL`	Bilirubin
`CHE`	Acetylcholinesterase
`CHOL`	Cholesterol
`CREA`	Creatinine
`GGT`	Gamma-Glutamyl Transferase
`PROT`	Proteins

Table 5.2: Tuning Methodology for each model.

Method	Description	No. of Cross Validations
`KNN`	Number of neighbors	10-Fold CV
`Naive Bayes`	Laplace Correction	10-Fold CV
`Random Forest`	MTRY, No. of Trees (fixed at 500)	10-Fold CV
`Logistic Regression`	-	10-Fold CV
`SVM Linear`	Cost, kernel (fixed at linear)	10-Fold CV
`LDA`	-	10-Fold CV
`QDA`	-	10-Fold CV
`Modified KNN`	N/A	2:1 CV

5.2 Results

In this study, the HCV dataset were split into approximately `2:1` for the training set and testing set, respectively. The model is built on the training dataset and later evaluated based on the testing dataset. To assess the performance of the seven existing classification models considered and the modified method in predicting Hepatitis C, the results of the model's performance metrics are evaluated and compared. Each model's testing accuracy, F_1 score, and AUC-ROC are displayed in Table 5.3.

5.2.1 F_1 Score

From the F_1 score perspective, there is no clear winner in terms of the models predictive performance. However, taking into consideration the purpose of this study, the Modified KNN method (97.8%) had a little predictive power over the existing KNN algorithm (96.5%). Other algorithms, including Random Forest, QDA, SVM Linear also performed well in predicting Hepatitis C using F_1 Score.

5.2.2 AUC-ROC

When we consider AUC-ROC as our performance indicator, the clear winner is the Modified KNN method, though, other existing methods like QDA, Random Forest and Naive Bayes performed well. Again, the Modified KNN with AUC of 92% has outperformed the existing KNN with AUC of 75.5%.

5.2.3 Accuracy

When we consider accuracy as our performance indicator, it is hard to determine the clear winner because Random Forest, QDA, and the Modified - all have 96.1% prediction accuracy in predicting Hepatitis C. Once again, the Modified KNN method has outperformed the existing KNN method (93.7%) in terms of prediction accuracy.

Table 5.3: Summary of Model's Performance Metrics based on real data.

Method	F_1 Score	AUC	Accuracy
`KNN`	0.9651	0.7554	0.9366
`Naive Bayes`	0.9725	0.8777	0.9512
`Random Forest`	0.9781	0.8831	0.9610
`Logistic Regression`	0.9704	0.7989	0.9463
`SVM Linear`	0.9756	0.8423	0.9561
`LDA`	0.9574	0.6902	0.9219
`QDA`	0.9780	0.9021	0.9610
`Modified KNN`	0.9779	0.9210	0.9610

Chapter 6

Conclusion

In this thesis, we have described some existing classification algorithms as well as our Modified KNN method for classification problem. In particular, we compared the Modified KNN method with the existing KNN algorithm as well as other classification methods – K-Nearest Neighbors (KNN), Naive Bayes algorithm, Random Forest, Support Vector Machine (SVM Linear), Logistic Regression (logit), Linear Discriminant Analysis, and Quadratic Discriminant Analysis. We compared the performance of these methods in term of F_1 score, AUC-ROC, and accuracy measures. The simulation study revealed that the Modified KNN method outperformed the existing KNN method when the simulated data contained different proportion of outliers (i.e., $\epsilon_i = 0, \quad 0.05, \quad \text{and} \quad 0.1$) across the performance metrics considered in this study. The Modified KNN actually predict class labels by fitting a linear model locally to the the training set indexed by the nearest neighbors. Furthermore, we applied our methods to HCV dataset which is available at UCI machine learning repository (*HCV data* 2020). We classified medical diagnosis of Hepatitis C. The Modified KNN method outperformed the existing KNN algorithm in predicting Hepatitis C patient. However, other classification methods like random forest, QDA and naive bayes algorithm performed well in predicting Hepatitis C. The performance evaluations confirm the validity of the Modified KNN method.

We take caution in making broad conclusion based on these results. For example, we cannot always guarantee that the Modified KNN method will consistently produce excellent performance. In particular, both simulated data and the HCV dataset features many predictor variables with different levels and different proportion of outliers. It would be interesting to observe how the results of methods considered in this thesis would vary

when applied to different datasets, particularly those featuring more outcome categories, numeric and categorical predictors, and different proportion of outliers.

Acknowledgment: I would like to express my sincere gratitude to the collaborators of Dr. Abhijit Mandal for providing me with their invaluable research proposal, which served as the foundation for my thesis. Their contribution to developing the main statistical algorithm (Modified KNN) and R code was critical to the success of this project. I do not claim any copywriting for the Modified KNN and the corresponding code, as it was solely created by Dr. Mandal and his team.

Bibliography

Aha, D. W. (1997), *Lazy learning*, Springer.

Bishop, C. M. & Nasrabadi, N. M. (2006), *Pattern recognition and machine learning*, Vol. 4, Springer.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**, 5–32.

Hastie, T., Tibshirani, R., Friedman, J. H. & Friedman, J. H. (2009), *The elements of statistical learning: data mining, inference, and prediction*, Vol. 2, Springer.

HCV data (2020), UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5D612.

Ho, T. K. (1995), Random decision forests, *in* 'Proceedings of 3rd international conference on document analysis and recognition', Vol. 1, IEEE, pp. 278–282.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013), *An introduction to statistical learning*, Vol. 112, Springer.

Jiang, Y. & Zhou, Z.-H. (2004), Editing training data for k nn classifiers with neural network ensemble, *in* 'Advances in Neural Networks–ISNN 2004: International Symposium on Neural Networks, Dalian, China, August 2004, Proceedings, Part I 1', Springer, pp. 356–361.

Keselj, V. (2009), 'Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6'.

Myers, R. H. & Montgomery, D. C. (1997), 'A tutorial on generalized linear models', *Journal of Quality Technology* **29**(3), 274–291.

Su, X. & Tsai, C.-L. (2011), 'Outlier detection', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1**(3), 261–268.

Whalen, S., Truty, R. M. & Pollard, K. S. (2016), 'Enhancer–promoter interactions are encoded by complex genomic signatures on looping chromatin', *Nature genetics* **48**(5), 488–496.

Wigglesworth, D. (2018), 'Comparing statistical classification techiques: An example with us census data'.

Appendix

R CODE

```
options(warn = -1)

#Loading packages
library(dplyr)
library(e1071)
library(caTools)
library(caret)
library(cvAUC)
library(MASS)
library(randomForest)
library(ModelMetrics)
library(robustbase)



##--------SIMULATION CODE-------##

generate_data <- function(n=50, beta0=c(0,1,1), epsilon = 0.05, muC = 5,
                          mu = 0, sigma = 1, vertical_outlier=T, ni = 1){

  #Inputes:
  #I: number of Binomial populations
```

```
# ni: paramaters of i-th Binomial population, currently a fixed value
#     for all samples (ni=1 indicates Bernoulli dist)
# k: number of explainatory variables, dimension of X is k+1 including
#    first column of one
# beta0: true value of the parameter (vector of length k+1)
# mu, sigma: xi ~ N(mu, sigma) iid for all i
# epsilon: proportion of contamination for x, i.e. leverage points
# muC: for leverage points xi ~ N(muC, 0.01*sigma)
# vertical_outlier: (T or F) leverage points are also vertical outliers,
#ie,  yi is fliped, ie yi = ni - yi
# n_iter: number of iteration

# initialize some values and containers
I <- n
k <- length(beta0) - 1 #number of predictors
Y <- vector("numeric", length = I)
# number of contaminated data for x (Leverage points)
I_cont = sum(runif(I)<epsilon)
# X: design matrix with explainatory variables, I*(k+1) matrix with first
#col 1 adds I_cont leverage points near muC

 X = cbind(rep(1,I), rbind(matrix(rnorm((I-I_cont)*k, mu, sigma), ncol=k),
                           matrix(rnorm(I_cont*k, muC, .01*sigma), ncol=k) ))



Xbeta0 = X %*% beta0
# pi_vec: vector containing pi_i
```

```
  pi_vec = exp(Xbeta0)/(1+exp(Xbeta0))
  #should we consider the cases where pi=0 or 1?

  ni_vec = rep(ni,I)
  for (y_indx in 1:I)
    Y[y_indx] = rbinom(1, ni_vec[y_indx], pi_vec[y_indx])

  # vertical outliers replaces yi to ni-yi
  if (I_cont>0 & vertical_outlier)
    Y[(I-I_cont+1):I] = ni_vec[(I-I_cont+1):I] - Y[(I-I_cont+1):I]

  list(X=X, Y=Y)

}
## Illustrations

# For k=2, n=50,
#data1 <- generate_data()

get_metrics <- function (yobs, y_pred) {

  yobs <- as.numeric(yobs)
  y_pred <- as.numeric(y_pred)



  # f1 score
  f1_score <- MLmetrics::F1_Score(yobs, y_pred,
                                  positive = NULL) #for binary class
```

```
  # AUC
  AUC_object <- ci.cvAUC(predictions = y_pred, labels= yobs,
                                    folds=1:NROW(yobs), confidence=0.95)
  auc <- AUC_object$cvAUC

  ACC = sum(yobs == y_pred)/length(yobs)


  return(c("f1score"=f1_score, "AUC"=auc,  ACC ))

}


#---------Modified KNN algorithm

euclidean_distance = function(a, b){
  #  We check that they have the same number of observation
  if(length(a) == length(b)){
    sqrt(sum((a-b)^2))
  } else{
    stop('Vectors must be of the same length')
  }
}


#Manhattan Distance
```

```
manhattan_distance = function(a, b){
  # We check that they have the same number of observation
  if(length(a) == length(b)){
    sum(abs(a-b))
  } else{
    stop('Vectors must be of the same length')
  }
}


#Cosine similarity

cos_similarity = function(a,b){
  if(length(a) == length(b)){
    num = sum(a *b, na.rm = T)
    den = sqrt(sum(a^2, na.rm = T)) * sqrt(sum(b^2, na.rm = T))
    result = num/den

    return(1-result)
  } else{
    stop('Vectors must be of the same length')
  }
}


#Minkowski Distance

minkowski_distance = function(a,b,p){
```

```
  if(p<=0){
    stop('p must be higher than 0')
  }

  if(length(a)== length(b)){
    sum(abs(a-b)^p)^(1/p)
  }else{
    stop('Vectors must be of the same length')

  }
}


#=====================
#k nearest neighbors
#=====================

nearest_neighbors = function(x,obs, k, FUN, p = NULL){

  # Check the number of observations is the same
  if(ncol(x) != ncol(obs)){
    stop('Data must have the same number of variables')
  }

  # Calculate distance, considering p for Minkowski
  if(is.null(p)){
    dist = apply(x,1, FUN,obs)
  }else{
```

```
    dist = apply(x,1, FUN,obs,p)
  }

  # Find closest neighbours
  distances = sort(dist)[1:k]
  neighbor_ind = which(dist %in% sort(dist)[1:k])

  if(length(neighbor_ind)!= k){
    warning(
      paste('Several variables with equal distance. Used k:',
      length(neighbor_ind)) )
  }

  out = list(neighbor_ind, distances)
  return(out)
}



#==============
#Prediction
#==============

knn_prediction = function(x,y, weights = NULL, x0, is.robust=FALSE){
  # here, y is supposed to be a column name (character)
  x = as.matrix(x)

  if(is.factor(x[,y]) | is.character(x[,y])){
```

```
  groups = table(x[,y])
  pred = names(groups[groups == max(groups)])
} else if(is.numeric(x[,y])){

  # Calculate weighted prediction
  if(!is.null(weights)){
    w = 1/weights/ sum(weights)
    pred = weighted.mean(x[,y], w)



    # Calculate standard prediction
  }else{
    if (sum(x[,y]) == nrow(x)) return(1)
    if (sum(x[,y]) == 0) return(0)

    # Robust Logistic Regression
    if (is.robust){
      y_fit = glmrob(as.formula(paste(y, "~ .")), data = data.frame(x),
                family = "binomial",
                control = glmrobMqle.control(acc = 1e-17,
                test.acc = "coef",  maxit = 100, tcc = 1.345))
    } else {
      y_fit = glm(as.formula(paste(y, "~ .")), data = data.frame(x),
                  family = "binomial", maxit = 50)
    }
    pred = predict(y_fit, newdata=data.frame(x0), type = "response")
    pred = ifelse(pred < 0.5, 0, 1)
```

```
      #Bianco-Yohai Estimator for Robust Logistic Regression
      # y_fit = BYlogreg(x[,(!colnames(x) %in% y)], x[,y])
      # pred = sum(y_fit$coefficients * c(1, as.matrix(x0)))
      # pred = 1/(1 + exp(-pred))
      # pred = ifelse(pred < 0.5, 0, 1)

      #For classical KNN
      #pred = mean(x[,y])
    }

  } else {
    stop('Y should be factor/character or numeric.')
  }

  return(pred)

}


#=============================================================================

knn_sc = function(x_fit, x_pred, y, k, is.robust=FALSE,
                  func = euclidean_distance,weighted_pred = F, p = NULL){

  # initializing predictions
  predictions = c()

  y_ind = which(colnames(x_pred) == y)
```

```
  for(i in 1:nrow(x_pred)){

    neighbors = nearest_neighbors(x_fit[,-y_ind],
                                  x_pred[i,-y_ind],k,FUN = func)

    if(weighted_pred){
      pred = knn_prediction(x_fit[neighbors[[1]], ],y, neighbors[[2]],
                            x0=x_pred[i,-y_ind], is.robust=is.robust)
    } else{
      pred = knn_prediction(x_fit[neighbors[[1]], ],y, x0=x_pred[i,-y_ind],
                            is.robust=is.robust)
    }

    # If more than 1 predictions, make prediction with 1 more k
    if(length(pred)>1){
      pred = knn(x_fit, x_pred[i,],y, k = k+1,
                 func = func, weighted_pred = weighted_pred, p == p)
    }

    predictions[i] = pred

  }
  return(predictions)
}
# varying sample sizes for different simulated data
# allow for varying percent outliers
```

```
#
### arguments
# n: sample size
#



my_sim <- function(n=100, percent_outlier=0.05, beta0=c(0,1,1), iter=10)
{
  # initialize containers for model metrics
  number_of_metrics <- 3
  knn_metrics <- nb_metrics <- RF_metrics <-
        glm_metrics <- matrix(nrow = iter, ncol = number_of_metrics)
  svm_metrics <- lda_metrics <- qda_metrics <- mod_knn_metrics <-
    mod_knn_rob_metrics <- knn_metrics

  for (i in 1:iter) {
    if (i%%10 == 0) cat(sprintf("Step %d/%d\n", i, iter))
    data <- generate_data(n=n, epsilon  = percent_outlier, beta0 = beta0)

    y <- factor(data$Y)
    X <- as.data.frame(data$X[,-1])
    n <- length(y)

    #set.seed(1234)
    test_ind <- sample(nrow(X), trunc((1/3)*nrow(X)), replace = FALSE)
    y_test <- y[test_ind]
    y_train <- y[-test_ind]
```

```
X_test <- X[test_ind, ]
X_train <- X[-test_ind, ]

#-----Fitting models

# Run algorithms using 10-fold cross validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"


#---------KNN Algorithm
fit.knn <- train(y=y_train, x=X_train, method="knn",
                 metric=metric ,trControl=trainControl)

y_pred <- predict(fit.knn, newdata = X_test)

knn_metrics[i,] <- get_metrics(y_test, y_pred)


#---------Naive Bayes
nb <- train(y=y_train, x=X_train, method="nb",
                    metric=metric ,trControl=trainControl)
y_pred <- predict(nb$finalModel,X_test)$class

nb_metrics[i, ] <- get_metrics(y_test, y_pred)


#---------Random Forest
```

```
fit.RF <- train(y=y_train, x=X_train, method="rf", ntree = 500,
                metric=metric ,trControl=trainControl)
y_probs <- predict(fit.RF, newdata=X_test, type="prob")[, 2]
y_pred <- factor(ifelse(y_probs >= 0.5, 1, 0))

RF_metrics[i, ] <- get_metrics(y_test, y_pred)


#------Logistic Regression

fit.glm <- train(y=y_train, x=X_train, method="glmnet",
                 family = "binomial",
                 metric=metric ,trControl=trainControl)
y_pred <- predict(fit.glm, newdata = X_test)
glm_metrics[i,] <- get_metrics(y_test, y_pred)



#---------Linear Discriminant Analysis
fit.LDA <- train(y = y_train, x=X_train, method="lda",
                 metric=metric ,trControl=trainControl)
y_pred <- predict(fit.LDA, newdata=X_test)
lda_metrics[i,] <- get_metrics(y_test, y_pred)


#-------Quadratic Discriminant Analysis
fit.QDA <-  train(y = y_train, x=X_train, method="qda",
                  metric=metric ,trControl=trainControl)
```

```
y_pred <- predict(fit.QDA, newdata=X_test)
qda_metrics[i,] <- get_metrics(y_test, y_pred)


#--------SVM Linear
svm_Linear <- suppressWarnings(train(y=y_train, X_train,
                                     method = "svmLinear",trControl=trainControl,
                                          preProcess = c("center", "scale")))
y_pred <- predict(svm_Linear, newdata = X_test)
svm_metrics[i,] <- get_metrics(y_test, y_pred)




Train <- cbind(as.numeric(y_train)-1,X_train)
colnames(Train)[1] <- "y"
Test <- cbind(as.numeric(y_test)-1, X_test)
colnames(Test)[1] <- "y"

#----------Modified KNN
y_pred <-  knn_sc(Train, Test, y= "y", k = 30, is.robust=FALSE)
y_pred <- factor(y_pred)

mod_knn_metrics[i,] <- get_metrics(y_test, y_pred)

#---------Robust Modified KNN
try({y_pred <-  knn_sc(Train, Test, y= "y", k = 30, is.robust=TRUE)
y_pred <- factor(y_pred)
```

```
  mod_knn_rob_metrics[i,] <- get_metrics(y_test, y_pred)}, silent = TRUE)
}



knn_metrics <- colMeans(knn_metrics)
nb_metrics <- colMeans(nb_metrics)
RF_metrics <- colMeans(RF_metrics)
glm_metrics <- colMeans(glm_metrics)
svm_metrics <- colMeans(svm_metrics)
lda_metrics <- colMeans(lda_metrics)
qda_metrics <- colMeans(qda_metrics)
mod_knn_metrics <- colMeans(mod_knn_metrics)
mod_knn_rob_metrics <- colMeans(mod_knn_rob_metrics)
method <- c("KNN", "Naive Bayes", "Random Forest", "Logistic",
            "SVM Linear", "LDA", "QDA", "Modified KNN",
                                    "Robust Modified KNN ")
metrics  <- rbind(knn_metrics, nb_metrics, RF_metrics, glm_metrics,
                  svm_metrics,lda_metrics,qda_metrics,
                  mod_knn_metrics, mod_knn_rob_metrics)

colnames(metrics) <- c("f1 score", "AUC", "Accuracy")
rownames(metrics) <- NULL
metrics <- data.frame(method, metrics)

return(metrics)
```

```
}


#----------displaying results

set.seed(12222)

#==== k=2, percent_outlier=0, n=100
 beta <- rnorm(3)
 results_2_0_100 <- my_sim(n=100, percent_outlier = 0, beta0 = beta,
                    iter = 100); results_2_0_100

#==== k=2, percent_outlier=0.05, n=100
 results_2_0.05_100 <- my_sim(n=100, percent_outlier = 0.05 ,beta0 = beta,
                   iter = 100); results_2_0.05_100

#==== k=2, percent_outlier=0.1, n=100
 results_2_0.1_100 <- my_sim(n=100, percent_outlier = 0.1 ,beta0 = beta,
                   iter = 100); results_2_0.1_100

#==== k=2, percent_outlier=0, n=200
 results_2_0_200 <- my_sim(n=200, percent_outlier = 0 ,beta0 = beta,
                   iter = 100); results_2_0_200

#==== k=2, percent_outlier=0, n=200
 results_2_0.05_200 <- my_sim(n=200, percent_outlier = 0.05 ,beta0 = beta,
                   iter = 100); results_2_0.05_200
#==== k=2, percent_outlier=0, n=200
```

```
results_2_0.1_200 <- my_sim(n=200, percent_outlier = 0.1 ,beta0 = beta,
                    iter = 100); results_2_0.1_200



#==== k=5, percent_outlier=0, n=100
beta <- rnorm(6)
results_5_0_100 <- my_sim(n=100, percent_outlier = 0, beta0 = beta,
                                iter = 100)

#==== k=5, percent_outlier=0.05, n=100
results_5_0.05_100 <- my_sim(n=100, percent_outlier = 0.05 ,beta0 = beta,
                                  iter = 100)

#==== k=5, percent_outlier=0.1, n=100
results_5_0.1_100 <- my_sim(n=100, percent_outlier = 0.1 ,beta0 = beta,
                                iter = 100)

#==== k=5, percent_outlier=0, n=200
results_5_0_200 <- my_sim(n=200, percent_outlier = 0 ,beta0 = beta,
                                iter = 100)

#==== k=5, percent_outlier=0, n=200
results_5_0.05_200 <- my_sim(n=200, percent_outlier = 0.05 ,beta0 = beta,
                                  iter = 100)
#==== k=5, percent_outlier=0, n=200
results_5_0.1_200 <- my_sim(n=200, percent_outlier = 0.1 ,beta0 = beta,
                                iter = 100)
```

```
#==== k=10, percent_outlier=0, n=100
beta <- rnorm(11)
results_10_0_100 <- my_sim(n=100, percent_outlier = 0, beta0 = beta,
                           iter = 100)

#==== k=10, percent_outlier=0.05, n=100
results_10_0.05_100 <- my_sim(n=100, percent_outlier = 0.05 ,beta0 = beta,
                              iter = 100)

#==== k=10, percent_outlier=0.1, n=100
results_10_0.1_100 <- my_sim(n=100, percent_outlier = 0.1 ,beta0 = beta,
                             iter = 100)

#==== k=10, percent_outlier=0, n=200
results_10_0_200 <- my_sim(n=200, percent_outlier = 0 ,beta0 = beta,
                           iter = 100)

#==== k=10, percent_outlier=0, n=200
results_10_0.05_200 <- my_sim(n=200, percent_outlier = 0.05 ,beta0 = beta,
                              iter = 100)
#==== k=10, percent_outlier=0, n=200
results_10_0.1_200 <- my_sim(n=200, percent_outlier = 0.1 ,beta0 = beta,
                             iter = 100)
```

Curriculum Vitae

Noah Owusu, born on October 22, 1991, is the third born child of Daniel Nyarko and Mercy Oppong. Upon completing high school in 2009, he enrolled at Accra Technical University formerly known as Accra Polytechnic in Ghana to pursue Higher National (HND) in Statistics in 2011. In 2014, he successfully graduated from Accra Technical University, ranking the best student of the Department of Statistics and eventually the best student of the school of Applied Sciences. As part of his national service in 2015, he worked as statistical assistant at UG college of Health Sciences, Korle Bu, Ghana. He continued to pursue Bachelor's of Science in Statistics at the University of Cape Coast (UCC), Ghana in 2018. He was adjudged the best student of Department of Statistics upon completion of his bachelors in 2020.

Noah's educational journey progressed in autumn of 2021 as he embarked on a Master's program in Statistics and Data Science at The University of Texas at El Paso, USA. This decision set the groundwork for his future ambitions of attaining a PhD in Statistics and establishing himself as a knowledgeable researcher and professional in the field. He actively participated in various workshops and seminars centered around statistics, data science, and mathematics. Simultaneously, he held a position as a Teaching and Research Assistant while pursuing his Master's degree. After completing his studies, Noah intends to pursue a doctoral degree in Statistics at Virginia Polytechnic Institute and State University (Virginia Tech).

E-mail address: *nowusu@miners.utep.edu*