2017

Brief Study of Classification Algorithms in Machine Learning

Ramesh Sankara Subbu

CUNY City College

Recommended Citation

http://academicworks.cuny.edu/cc_etds_theses/679

How does access to this work benefit you? Let us know!

Follow this and additional works at: http://academicworks.cuny.edu/cc_etds_theses

Part of the Other Computer Engineering Commons, and the Other Electrical and Computer Engineering Commons

This Thesis is brought to you for free and open access by the City College of New York at CUNY Academic Works. It has been accepted for inclusion in Master’s Theses by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.
Brief Study of Classification Algorithms in Machine Learning

EE I9900 - Master’s Thesis
Submitted in partial fulfillment of the requirement for the degree
Master of Engineering (Electrical)
Spring 2017
At
The City College of New York
Of the
City University of New York

By
Ramesh Sankara Subbu

Approved:

Professor Bo Yuan, Thesis Advisor

Professor Roger Dorsinville, Chair
Department of Electrical Engineering
## Contents

1 Introduction 1

2 Overview of Machine Learning 5

3 Types of Machine Learning Algorithms 8

4 Steps in developing a Machine Learning Algorithm 11

5 Supervised Learning 13

6 k-Nearest Neighbors 16

6.1 Background 17

6.2 Flowchart 19

6.3 Example with Python Code 21

6.4 Explanation 25

6.5 Results 27

7 Decision Trees 32

7.1 Background 33

7.2 Flowchart 35

7.3 Example with Python Code 37

7.4 Explanation 43

7.5 Results 46

8 Naïve Bayes 47

8.1 Background 48

8.2 Example with Python Code 50

8.3 Explanation 54

8.4 Results 56

9 Conclusion 57

10 Acknowledgements 58

11 References 59
1. Introduction

Machine Learning is the study and construction of algorithms that can gain insight from sample dataset and make data-driven predictions or decisions on new data. Tom M. Mitchell provided a formal definition: "A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$ if its performance at tasks in $T$, as measured by $P$, improves with experience $E$"[1]. It involves development of computer programs which changes or learns when exposed to new data which is like data mining. Both systems search through data to look for patterns. However, data mining extracts data for human comprehension whereas machine learning uses that data to detect patterns in data and adjust program actions accordingly.

Machine learning is done always based on observations or data, direct experience, or instruction. So, in general, machine learning is about learning to do better in the future based on what was experienced in the past. The goal is to devise learning algorithms that do the learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as “programming by example.” Often we have a specific task in mind, such as spam filtering. But rather than programming the computer to solve the task directly, in machine learning, we seek methods by which the computer will come up with its own program based on examples that we provide. Machine learning is a core subarea of artificial intelligence. It is very unlikely that we will be able to build any kind of intelligent system capable of performing complex tasks such as language or vision, without using learning to get there. These tasks are otherwise simply too difficult to solve. Further, we would not consider a system to be truly intelligent if it were incapable of learning since learning is at the core of intelligence. Although a subarea of
AI, machine learning also intersects broadly with other fields, especially statistics, but also mathematics, physics, theoretical computer science and more.

Machine learning is nowadays used in every industry to solve their common problems, some of them are explained below:

- Optical Character Recognition (OCR): conversion of handwritten or printed characters in images into machine-encoded text
- Face detection: find faces in images or videos using image processing
- Spam filtering: identify email messages as spam or non-spam
- Topic spotting: categorize news articles into generics such as politics, sports, entertainment, etc.
- Spoken language understanding: within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories
- Medical diagnosis: diagnose a patient as a sufferer or non-sufferer of some disease
- Predictive Modeling: target specific customers and improve products marketing process
- Fraud detection: identify fraudulent credit card transactions
- Weather prediction: predicting the probability of getting rain or snow

The primary goal of machine learning research is to develop general purpose algorithms of practical value with efficiency. In the context of learning, we should care about the amount of data that is required by the learning algorithm in addition to time and space efficiency. Learning algorithms should serve a general purpose in solving problems
that can be easily applied to a broad class of learning problems, such as those listed above. Of primary importance, we want the result of learning to be a prediction rule which should be accurate in making predictions on a new data. Occasionally, we may also be interested in the interpretability of the prediction rules produced by learning. As mentioned above, machine learning can be thought of as “programming by example.” The major advantage of machine learning over static programming is the results are often more accurate with machine learning than static programming results because the machine learning algorithms are data driven, and can examine large amounts of data. On the other hand, a human expert who writes static programs is likely to be guided by imprecise impressions or perhaps an examination of only a relatively small number of examples or data. Figure 1 show the general process involved in a typical machine learning model.

![Diagram of a general Machine Learning Process](image)

*Figure 1. Diagram of a general Machine Learning Process*
For instance, it is easy for humans to label images of letters by the character represented, but we would have trouble in explaining how we did it in precise terms. Another reason to study machine learning is the hope that it will provide insights into the general phenomenon of learning. Some of the details we might learn are the intrinsic properties of a given learning problem that makes it hard or easy to solve and know ahead of time about what is being learned to be able to learn it effectively. In this report, we are interested in designing machine learning algorithms, but we also hope to analyze them mathematically to understand their efficiency. Through theory, we hope to understand the intrinsic difficulty of a given learning problem and we attempt to explain phenomena observed in actual experiments with learning algorithms.
2. Overview of Machine Learning

Machine learning is basically turning data into information. The knowledge or insight we try to learn from raw data cannot be done by just looking at it, for example, a spam email cannot be detected by looking at the occurrence of a single word but rather looking at certain words occurring in combination or length of the email and other such factors can help you in detecting it. Machine learning also makes use of statistics and it can be used to solve any problems that needs to interpret and act on data later use those facts learned to decide on a new set of data. Usually static programs are used to solve deterministic problem with a definite solution but problems that are not deterministic which doesn’t provide enough data about it we take the approach called machine learning. In the early days, it was difficult to make realistic decisions using machine learning due to inadequate datasets to train the algorithms. But due to resurgence in sensors and their ability to connect to Internet, nowadays the real problem is to efficiently sort through the available abundant free data and make use them to train machine learning algorithms. The increase in smartphones usage which contains various sensors such as accelerometers, GPS, temperature sensors within it has also added fuel to the increase in the data collection. The current development trends in mobile computing and Internet of Things will lead to the generation of more and more useful data in the future. Since lots of economic activities are dependent on data we cannot afford to get lost in them, so machine learning helps to get through these data and extract important information from them.

Let’s explain the key terminologies involved in machine learning using an example before we get into actual algorithms. Consider we are building a coin
classification system which can be used to count different coins ranging from 1 cent to 1 dollar. By creating a computer program, we have replaced a human being to count coins. Each coin has its own characteristics such as diameter, thickness, mass and edge which are called as features or attributes and the corresponding value is called as target variable as shown in Table 1.

<table>
<thead>
<tr>
<th>S. No</th>
<th>Diameter</th>
<th>Thickness</th>
<th>Mass</th>
<th>Plain Edge</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.05 mm</td>
<td>1.55 mm</td>
<td>2.50 g</td>
<td>Yes</td>
<td>1¢</td>
</tr>
<tr>
<td>2</td>
<td>21.209 mm</td>
<td>1.95 mm</td>
<td>5 g</td>
<td>Yes</td>
<td>5¢</td>
</tr>
<tr>
<td>3</td>
<td>17.907 mm</td>
<td>1.35 mm</td>
<td>2.268 g</td>
<td>No</td>
<td>10¢</td>
</tr>
<tr>
<td>4</td>
<td>24.257 mm</td>
<td>1.75 mm</td>
<td>5.67 g</td>
<td>No</td>
<td>25¢</td>
</tr>
<tr>
<td>5</td>
<td>30.607 mm</td>
<td>2.15 mm</td>
<td>11.34 g</td>
<td>No</td>
<td>50¢</td>
</tr>
<tr>
<td>6</td>
<td>26.492 mm</td>
<td>2.00 mm</td>
<td>8.10 g</td>
<td>Yes</td>
<td>1$</td>
</tr>
</tbody>
</table>

*Table 1. Coin Classification based on four features*

The first three features are numeric which takes a decimal value whereas the plain edge feature takes a Boolean value 1 or 0. Classification is one major task in machine learning and here each coin is classified into its own value using combination of data from image processing and other sensors, so that it can be counted to a total value. This report majorly revolves around classification algorithms. Once the classification algorithm to be used is finalized, we should train the algorithm by feeding it with quality data (training examples) called as training set. In table 1 there are 6 training examples with each has 4 features and 1 target variable. Machine learning algorithm learns some relationship between features and target variable and it will try to predict the target
variable for the new data. In this example target variables are just the value of coins so every time when a new coin comes inside the machine, it will measure its features and predict the value of the coin and these new measured features are called as test set. Accuracy of the algorithm can be calculated by comparing the actual value of the coin with the predicted target variable of the algorithm. The process of viewing the data learned by the algorithm is called as knowledge representation which can be a set of rules, a probability distribution function or an example from the training set.
3. Types of Machine Learning Algorithms

Machine learning algorithm can model each problem differently based on the input data, so before getting into algorithms we should briefly view various kinds of learning styles broadly used. This way of organizing machine learning algorithms forces us to choose the right algorithm to tackle a given problem based on the available input dataset and model preparation process and achieve efficient results. We can divide machine learning algorithms into three different groups based on their learning style:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

SUPERVISED LEARNING

*Supervised learning* occurs when an algorithm learns from input data also known as training data which has known target responses or labels that can be a numeric value or string. A model is prepared through the training or learning process which predicts the correct response when given a new example. The supervised approach is further divided into two: *Classification* and *Regression*. In classification, the algorithm predicts the class to which the given test data fall into whereas regression predicts a numeric value for target variable. For example, we might treat investment as a classification problem (will the stock go up or down) or a regression problem (how much will the stock go up)
through which we want the computer to learn directly how to decide to make investments to maximize wealth.

UNSUPERVISED LEARNING

Unsupervised learning occurs when an algorithm learns from input data without any labels and does not have a definite result, leaving the algorithm to determine the data patterns on its own. A model is prepared by learning the features present in the input data to extract general rules. It is done through a mathematical process to reduce redundancy or to organize data by similarity. Unsupervised learning is again majorly used in two different formats: Clustering in which we group similar items together and density estimation which is used to find statistical values that describe the date. For example, customer targeted online advertisements are based on this learning model which derives its suggestion form your past purchases. The recommendations are based on an estimation of what group of customers you resemble the most and then inferring your likely preferences based on that group.

REINFORCEMENT LEARNING

Reinforcement learning allows machines to determine automatically its behavior within a specific context to maximize its performance. Simple reward feedback is required to learn its behavior known as reinforcement signal. This learning occurs when you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can accompany an example with positive or negative feedback per the solution the algorithm proposes. Reinforcement learning is connected to applications for which the algorithm must make decisions unlike unsupervised learning and the decisions
bear consequences in real world. It is considered like learning by trial and error method. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves. In this case, an application presents the algorithm with examples of specific situations, such as set of moves in a chess game. The application lets the algorithm know the outcome of actions it takes, and learning occurs while trying to avoid checkmate. This learning is a steady improvement process and the chess algorithm improves its mastery based on number of games it played and level of difficulty it came across.
4. Steps in developing a Machine Learning Algorithm

There are six general steps we would follow throughout this report while implementing machine learning algorithms in the forthcoming sections which are briefly explained below:

1. **Collect data**: Data collection is a tedious process and it can be done by scraping some websites and extracting from them; get information from RSS feed or collect reading from any sensors and internet of things. To keep the process simple, we made use of publicly available data for this thesis.

2. **Prepare the input data**: Once the data is available convert them into a format which your algorithm accepts which will help us to use the same set of information with various algorithms. But the algorithm-specific formatting is usually trivial compared to collecting data.

3. **Analyze the input data**: This is looking at the data from the previous task to make sure the text file created from steps 1 & 2 are valid working data which matches our expectation. We can also search for any recognizable patterns also can plot the data in two or three dimensions for deep analysis. When there are multiple features available in data we can reduce them to three important features for plotting purpose.

4. **Train the algorithm**: This step can also be called as learning process. The combination of training and testing the algorithm is the “core” of any machine learning process. We would feed the algorithm with valid analyzed data called as training set and extract knowledge or information. This knowledge you often store in a format that’s readily useable by a machine for the next two steps. In the case
of unsupervised learning, there’s no training step because you don’t have a target value. Everything is used in the next step.

5. **Test the algorithm:** Information learned from training process is used here. When evaluating an algorithm, testing needs to be done to determine its accuracy. In the case of supervised learning, the target variables are known for test data which is used to evaluate the algorithm. In unsupervised learning, you may have to use some other metrics to evaluate the success. In either case, if the efficiency is not satisfactory we have to go back to step 4 redo the learning process using different and more accurate training data and test the algorithm again.

6. **Use it:** We make use of algorithm to make decisions or predict a solution at this step. If you are not satisfied with the accuracy revisit the process from initial step and retrain using more data as Machine Learning is a continuous development process.

Even though these six steps hold perfectly for creating a machine learning algorithm, for each algorithm based on the problem to be solved small changes or few steps should be added in-between them to facilitate data as per the requirement. As mentioned this paper focuses majorly on classification algorithms in supervised learning our next part explains clearly the total process involved in creating a supervised learning model.
5. Supervised Learning

Let us start this section with a detailed workflow diagram of supervised learning for both classification and regression which differs only in the target variable predicted (class or decimal) as in Figure 2.

Figure 2. Workflow diagram of Supervised Learning Process
To explain the workflow diagram in Figure 2 we will use the famous “Iris” flower dataset [3] example which is most commonly used to explain various concepts in data science here it is used to explain supervised machine learning task. This dataset has four features: Sepal width, sepal length, petal width and petal height which falls into three flower species also called class labels: setosa, virginica and versicolor. If iris dataset consists of series of images of flowers which is considered the raw data, the second step of pre-processing must be done which is feature extraction to measure the four features in centimeters. We cannot afford to have samples with missing data so if the data sparsity is less we could remove the samples with missing value from dataset or replace the missing values using some statistics instead of removing it. Third step sampling is the process of splitting randomly our dataset into training and test dataset. The training dataset is used to train the algorithm whereas test dataset is to evaluate the efficiency of the algorithm at the end. Next process called cross-validation is used to evaluate different combinations of feature selection, dimensionality reduction and learning algorithms. The common one used is k-fold cross-validation in which the training dataset is split into k subsets (k-1 subset is used for training and 1 subset will be used for testing), this splitting can help in calculating the average error rate once the learning process is done. Normalization is done to give equal importance to every feature in the dataset since each feature can have different range of values while learning and making decisions, it must be done on both training and test data. There are many kinds of learning algorithms in this paper we would explain about k-Nearest Neighbor, Decision trees and Naïve Bayes in the later sections. In post processing process, we would evaluate accuracy of the algorithm by testing it using the test data and if accuracy doesn’t satisfy the expected process we can
always restart the process by providing the algorithm with more accurate and abundant data. We can also refine our input dataset collection and preparation technique to achieve better result. When the algorithm achieves the expected accuracy, we can use the algorithm to predict real data.

Classification is a part of supervised learning model where the algorithm predicts the class under which the new data falls where the class should not be a numeric value. In this paper, we are going to deal with three important classification algorithms:

- k-Nearest Neighbor
- Decision Trees
- Naïve Bayes
6. k-Nearest Neighbor

In this section, we will discuss about our first classification algorithm: k-Nearest Neighbors. It is simple to understand and easy to implement compared to other machine learning algorithms. This section will start with an explanation about basic working concept behind this algorithm followed by a flow chart which explains step wise process involved. To explain the algorithm, we would state an example for improving results from a dating website and the corresponding python script used to implement it. The python script would be clearly explained function wise in the next subsection followed by the results obtained from that code.

The advantages in using this algorithm are its high accuracy, insensitive to outliers and no assumption about data. It has its own disadvantages also which are its requirements for lot of memory and expensive computation. This algorithm works with both numeric and nominal values.
6.1. Background

The k-Nearest Neighbor (kNN) algorithm is one of the most widely used classification algorithm due to its simplicity and easy implementation. It is also used as the baseline classifier in many domain problems [4]. kNN algorithm is a conventional non-parametric classifier [5] usually used for classification and regression problems. We start with a set of data each with a data point and a known class and these are divided into two subsets called as training data and test data. The process of predicting the class for new data based on the classes of available training data is called as classification problem. kNN is a type of lazy learning method because we don’t need to train the algorithm but during the classification phase it goes through all the training data to calculate the distance between them and input data and predicts the class of input data [1].

The distance between two points is decided by a similarity measure called as Euclidean distance, even though there are many other ways to measure the distance Euclidean distance measurement is commonly used since there is no comparative study examining the effect of distance function over the efficiency of kNN. Mathematical formula for measuring Euclidean distance between two points p & q with n elements is given below:
Once the distance between the input data and the training data are measured using the above formula, then $k$ number of nearest points to the input data are selected and majority class of the selected neighbors will become the predicted class for new data. Hence the name $k$-Nearest Neighbors. Euclidean distance calculations hold good for categorical and numerical datasets but not for the mixed type of datasets [6] & [7].
6.2. Flowchart

In k-Nearest Neighbors (kNN) algorithm we would have training dataset and test dataset. Each instance of training data would have several features and one label, since we are discussing about single label classification only. So, we know the labels to which each instance of data falls into. The whole purpose for developing this algorithm is to identify the label for the new data given without a label of its own and the whole process involved is explained in the workflow diagram given in Figure 3.

![Workflow diagram of k-Nearest Neighbor Algorithm](image)

*Figure 3. Workflow diagram of k-Nearest Neighbor Algorithm*
We start the process by initializing the value $k$ (integer) whose importance will be explained later. In the next step, we would compute the Euclidean distance between the new input sample and every training sample provided to the algorithm. It is followed by sorting out the distance of each training sample from the input data and once the distances are sorted we choose $k$ nearest neighbors to the input data, this is where $k$ becomes useful. Once the neighbors are chosen, new input data is given the label which is a majority among its neighbors. This is the simple process behind the powerful and most common classifier used among machine learning models.
6.3. Example with Python Code

To explain effectively the k-Nearest Neighbors algorithm we are going to look at an interesting example of filtering the matches recommended by a dating site per the user’s input and dividing the recommendation into three categories: doesn’t like, like in small doses and like in large doses. The user’s input is derived from her past dating experiences with the persons recommended by the site. The data was collected in a text file by the user which contains percentage of time spent in playing videogames, liters of ice-cream consumed weekly and number of frequent flyer miles earned per year as three features used to determine the likability of the person. We would prepare the data by parsing the text file in python. Analysis of data is done by Matplotlib to make 2D plots of data. Algorithm training is not needed in kNN since the Euclidean distance between the input sample and training sample is done every time. Then we would test the algorithm by comparing the result of test data from algorithm to its actual result which would give us the error rate. At last create a program where the user would receive the predicted output (like or dislike) by feeding few inputs. To implement the algorithm, we have used an open data science platform called Anaconda powered by python and it has inbuilt NumPy and Matplotlib packages which makes our life easy. Let us look at the python code designed to implement kNN algorithm for this example.
```python
from numpy import *
import operator

# Python code

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndices = distances.argsort()
    classCount = {}
    for i in range(k):
        voteILabel = labels[sortedDistIndices[i]]
        classCount[voteILabel] = classCount.get(voteILabel, 0) + 1
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

# Python code

def file2matrix(filename):
    love_dictionary = {'largeDoses': 3, 'smallDoses': 2, 'didntLike': 1}
    fr = open(filename)
    arrayOLines = fr.readlines()
    numberOfLines = len(arrayOLines)
    returnMat = zeros((numberOfLines, 3))
    classLabelVector = []
    index = 0
    for line in arrayOLines:
        line = line.strip()
        listFromLine = line.split('	')
        returnMat[index,:] = listFromLine[0:3]
        if listFromLine[-1].isdigit():
            classLabelVector.append(int(listFromLine[-1]))
        else:
            classLabelVector.append(love_dictionary.get(listFromLine[-1]))
        index += 1
    return returnMat, classLabelVector
```

22
def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals, (m,1))
    normDataSet = normDataSet/tile(ranges, (m,1)) #element wise divide
    return normDataSet, ranges, minVals

def datingClassTest():
    hoRatio = 0.50  #hold out 10%
    datingDataMat,datingLabels = file2matrix('datingTestSet.txt')  #load data set from file
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],
                                      datingLabels[numTestVecs:m,3])
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult, datingLabels[i])
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print "the total error rate is: %f" % (errorCount/float(numTestVecs))
    print errorCount
def classifyPerson():
    resultList = ['not at all', 'in small doses', 'in large doses']
    percentTats = float(raw_input("percentage of time spent playing video games?"))
    ffMiles = float(raw_input("frequent flier miles earned per year?"))
    iceCream = float(raw_input("liters of ice cream consumed per year?"))
    datingDataMat, datingLabels = file2matrix('datingTestSet.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    inArr = array([ffMiles, percentTats, iceCream, ])
    classifierResult = classify0((inArr - 
                    minVals)/ranges, normMat, datinglabels, 3)
    print "You will probably like this person: %s" % resultList[classifierResult - 1]

print classifyPerson()
6.4. Explanation

In the python code, we first import NumPy which is our scientific computing package and then the operator module which is later used in the algorithm for sorting. The idea behind our first function `classify0( )` is as follows:

- Calculate the Euclidian distance between input data (inX) and a training data
- Sort the distances in ascending order along with the distances of other training data
- Take k lowest distances from sorted distance data
- Choose the majority class among k lowest distances
- Return the majority class as predicted class of input data

Next our function `file2matrix( )` works basically to prepare a file i.e. parsing data from a text file to python. This function briefly does the following functions:

- Reads the file and counts the number of lines present in text file
- Create a NumPy matric to populate and return
- Loop all the lines and strip the return line character using `line.strip( )` and use tab delimiter between them
- Return the first three elements of each line as `returnMat` which are the features of dataset and the fourth element of the line as `classLabelVectors` which is the class

Let us now talk about `autoNorm( )` function which is used to normalize each value with respect to 1. In this example, frequent flier miles will always dominate the Euclidean distance outcome irrespective of liters of ice-cream and percentage of time spent playing
video games differences because of its high value. But the user treats all three features equally when deciding the likability of the person so to make the impact of each features on Euclidean distance equal we normalized them. This function is based on the following idea:

- Get the minimum and maximum values of each column and place it in minVals & maxVals
- Perform element-wise calculation for normalized value with the help of following formula:

\[
\text{Normalized value} = \frac{\text{old value} - \text{min}}{\text{max} - \text{min}}
\]

The function `datingClassTest( )` calls for two functions `file2matrix( )` and `autoNorm( )` which opens the input data parse them into python and normalize each values of functions. Then this function splits the input dataset into two separate datasets called as training dataset and testing dataset. These two datasets are fed into `classify0( )` function and the returned matrices from it are used to display the comparison between original class of test dataset and the class predicted by the algorithm. Also, the corresponding error rate is also displayed by this function.

The final `classifyPerson( )` function uses the algorithm to predict totally new data without any class. This function would ask for the input features from the user and predict the output class by calling for `file2matrix( )`, `autoNorm( )` and `classify0( )` functions in order. The output has been predicted with the error rate of 0.064000 for the given training data which is shown in the following section.
6.5. Results

We would start this section by showing 2D plots created by Matplotlib during the input data analysis phase. The python code used to create the plot is given below:

```python
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(datingDataMat[:,0], datingDataMat[:,2], 15.0*array(datingLabels), 15.0*array(datingLabels))
plt.show()
```

The input features number of frequent flyer miles/year, percentage of time spent playing video games and liters of ice-cream consumed weekly are represented as columns 0,1 & 2 respectively while plotting as given in input text file. Figure 4 represents the plot where x axis represents frequent flyer miles and y axis shows time spent playing video games. The violet color plot shows the “did not like” class; yellow color shows “liked in large doses” class and blue color shows “liked in small doses” class. The color representation remains the same for Figure 5 & 6 too where only the x and y axis representation changes.
Figure 4. Frequent flyer miles earned yearly vs time spent playing video games

Figure 5. Time spent playing video games vs liters of ice-cream consumed weekly
Figure 6. Frequent flyer miles earned vs liters of ice-cream consumed weekly
Figure 7 would show us the error rate which affected the efficiency of the kNN algorithm during its implementation along with the error rate and few examples showing the expected test data class and the actual test data class predicted by the algorithm. The arrow mark on the left-hand side of the figure shows the typical error in the testing phase. The error rate is basically the error count divided by the total number of test data given to the algorithm. In this example, the error rate is 0.064000 with an error count of 32.

![Python console]

```
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 2, the real answer is: 2
32.0
None
```

Figure 7. Screenshot of testing phase showing an error and its error rate & count
We would finish the results subsection with the screenshot in Figure 8 showing the final prediction output when new features were fed to the algorithm by the user.

![Screenshot of output predicted when new features were fed](image)

*Figure 8. Screenshot of output predicted when new features were fed*
7. Decision Trees

In this section, we will discuss about our next classification algorithm: Decision Trees. It is the most commonly used machine learning technique. This section will start with an explanation about background of this algorithm followed by a flow chart which explains step wise process involved. To explain the algorithm, we would state an example to predict the contact lens type people would need and the corresponding python script used to implement it. The python script would be clearly explained function wise in the next subsection followed by the results obtained from that code.

The major advantage of decision tree is that humans can easily understand the data and they are computationally cheap to use. Decision trees still holds good for missing values and can deal with irrelevant features. The biggest disadvantage is they are prone to overfitting. This algorithm also works with both numeric and nomial values like kNN.
7.1. Background

A decision tree is a representation of a design process to determine the class of a given feature. Each node of the tree can be either a leaf node (or answer node) that contains a class name or a non-leaf node (decision node) that contains an attribute test with a branch to another decision tree for each possible value of the attribute. Generally, in a decision tree plot a leaf node will be in the shape of oval whereas decision nodes would be a rectangle with arrows pointing the connections between appropriate nodes [8]. The core strength of every machine learning model is based on its underlying learning strategies and the strategy behind this implemented decision tree is ID3 algorithm [9] which takes care of splitting the dataset based on the attribute and the right place to stop splitting. Those processes are explained in the following sections.

Before we move on the next section, let us discuss about the mathematical calculations and theories surrounding the ID3 algorithm. To split the dataset based on best attribute we use the concepts from Information theory formulated by Shannon [10]. Using information theory, we would measure the difference in information before and after the split called as information gain. The measure of information of a set is called as Shannon Entropy or Entropy which is defined as:

\[
H = -\sum_{i=1}^{n} p_i \left(\log_2 p_i \right)
\]

\[
p_i = \text{Probability of choosing class } i
\]

\[n = \text{number of classes}\]
Higher the entropy, more mixed is the dataset. So, the difference in Entropy before and after the split is called as Information gain and the split with the highest information gain will be considered as the best feature to split after all Decision tree algorithm is performed for classifying data instances belonging to same class to the same leaf node.
7.2. Flowchart

Decision tree algorithm falls under supervised learning technique and is the most commonly used technique. We will follow ID3 algorithm which decides the best feature to split and indicates the time to stop splitting the tree algorithm. The structured workflow followed in implementing the decision tree algorithm is given below in Figure 9.

![Flowchart Diagram]

*Figure 9. Workflow Diagram of Decision Tree using ID3 algorithm*
Deciding on the best feature to split and the actual splitting process is done using ID3 algorithm and the process of calculating the information of a dataset is defined as Shannon Entropy or Entropy. From the workflow diagram, we infer the following steps:

- Start the algorithm by collecting the input data and preparing it for further processing
- Use the input data as training dataset
- Decide the best feature to split based on calculating the information gain, higher the information gain best is the feature to split
- Split the dataset into subsets based on the best feature
- Check whether all the data in the subset belongs to the same class, if yes stop the splitting process. If there are different classes available restart deciding the feature to split on and split the dataset further leading to different branches.

Hence the process continues until every end node of the tree has elements belonging to the same class. Next section explains the example used along with the python code used to implement it.
7.3. **Example with Python Code**

We would look at an example that predicts the contacts lens type that needs to be prescribed based on the given dataset. From the results, we can get an insight about the process by which the doctor prescribes the contacts lens to the patient. The data was collected in the text file which is provided to us and we are not interested in data collection process now. The collected data is prepared by parsing it into python using tab delimited lines. Analysis phase is done by reviewing the data visually and creating a tree plot finally. We train the algorithm by creating a tree data structure and test it for errors. We can use the data structure code for different scenarios by providing a different training data and usually decision trees are used for better visual understanding of data by humans.
```python
from math import log
import operator
import matplotlib.pyplot as plt

def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2)
    return shannonEnt

def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet
```
```python
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature

def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

39
def createTree(dataSet, labels):
   classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)
    return myTree

decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrows_args = dict(arrowsstyle="<>")

def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',
    xytext=centerPt, textcoords='axes fraction',
    va="center", ha="center", bbox=nodeType, arrowprops=arrow_args )
def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = myTree.keys()[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            numLeafs += getNumLeafs(secondDict[key])
        else:
            numLeafs +=1
    return numLeafs

def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = myTree.keys()[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    return maxDepth

def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)
def plotTree(myTree, parentPt, nodeTxt):
    numLeafs = getNumLeafs(myTree)
    getTreeDepth(myTree)
    firstStr = myTree.keys()[0]
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            plotTree(secondDict[key], cntrPt, str(key))
        else:
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
            plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD

def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])  # createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0;
    plotTree(inTree, (0.5,1.0), '')
    plt.show()
7.4. **Explanation**

This section would explain the implemented python code function wise for the given example. To start the code, we import logarithmic tool from math module, operator module and Matplotlib module for plotting the decision tree. Our first function `calcShannonEnt()` is useful in calculating the information of a dataset using Shannon Entropy with the following steps:

- Calculate number of instances in dataset and create a dictionary which counts all possible classes and its total number of occurrence
- Use the frequency of all labels to calculate its probability
- Calculate Shannon entropy by implementing its formula and use the above calculated probabilities

Next function `splitDataSet()` is used to split the dataset which is done by creating a separate list for saving the new dataset. Then this function would cut out input dataset based on a feature.

Now `chooseBestFeatureToSplit()` function will analyze the best feature to split such that the information gain would be large i.e. difference between the old entropy and new entropy is large because higher the entropy more messier is the data. It is carried out in following ways:

- Calculate the Shannon entropy of the input dataset before splitting by calling `calShannonEnt()` function
- Create a unique list of values for each feature from the dataset using native set data type
• Now split the dataset based on each unique feature in the set, their corresponding entropies are calculated and summed together
• Calculate the information gain, find the largest information gain i.e. largest loss in entropy and return the best feature

Next the \textit{majorityCnt( )} function creates a dictionary with unique class names in \textit{classList} and the object of the dictionary is the frequency of occurrence of each class label in the list. Finally use the operator module to sort the dictionary and returns the class with the greatest frequency. It is useful when the dataset has no more attributes left for splitting but still the classes are not the same.

The tree building code \textit{createTree( )} gets two inputs dataset and a list of labels. This function first creates a list with labels from each of the features in the given dataset followed by two stopping condition one when all the classes are equal and if there are no more features available to split. If the stopping conditions are not met, the function calls for \textit{chooseBestFeatureToSplit( )} function to choose the best feature. Get all the unique values from the given dataset for the best feature which is stored in a set data type. At last we would recursively call for \textit{createTree( )} function for the new datatype until the stopping conditions are met. Hence, the Tree is created for the given dataset.

We would now start the second portion of our code for creating a plot i.e. provide a visual result for decision tree algorithm to make the user understand it more easily and effectively. Now let us start import Matplotlib module into our code before staring the functions. We would first define some constants which would useful in box and arrow formatting in the tree plot later. Then create a \textit{plotNode( )} function which is useful in drawing annotation with arrows.
Before plotting the tree, we should know the number of leaf nodes and number of levels the tree travel which would be helpful in sizing X & Y direction properly. It was achieved using functions. Next function) is used to calculate the midpoint between the parent and child and place a small text label there.

Now the most important function for plotting a tree `plotTree()` comes which follows the steps:

- Calculate the width and height of the tree to place the leaf nodes and decision nodes at right places by calling `getNumLeafs()` & `getTreeDepth()` functions

- Plot the value for the feature along the split exactly at the center of the arrow by calling `plotMidText()` function

- Decrement the Y axis value when you are about to plot the child node, follow these steps recursively until every leaf node is plotted

The last function `createPlot()` handles setting up the overall image, calculates the global tree size and kicking start the `plotTree()` function and the other mentioned functions follows recursively.
7.5. Results

We would code few lines to feed the input text file and get the tree output by calling `createTree( )` and `createPlot( )` functions also giving all the available labels as input to these functions.

```python
fr = open('lenses.txt')
lenses = [inst.strip().split(' \t') for inst in fr.readlines()]
lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
lensesTree = createTree(lenses, lensesLabels)
print lensesTree
lensesPlot = createPlot(lensesTree)
```

The output Tree and the output plot received as output for the given example is shown in Figure 10.

![Figure 10. Output Tree & Plot of Decision Tree Algorithm](image-url)
8. Naïve Bayes

In this section, we will discuss about our next classification algorithm: Naïve Bayes. From the previous two classification algorithms, we estimated a definite class for the input data and then we calculated the error rate for the same. But in Naïve Bayes we would guess the best class and assign a probability for that best guess. This section will start with an explanation about background of this algorithm followed by an example to classify the spam emails and the corresponding python script used to implement it. The python script would be clearly explained function wise in the next subsection followed by the results obtained from that code.

The major advantage of decision tree is it works with a small amount of data and handles multiple classes. But it is sensitive to the input data preparation process. This algorithm will work with nominal values only.
8.1. Background

Naïve Bayes is a simple form of Bayesian classifiers, based on Bayes Decision theory. Bayes’ theorem plays a major role in the classification process which is explained in the next section. Bayesian classifiers assigns the most likely class to a given instance. The Naïve Bayes classifier assumes that the effect of an attribute on a class is statistically independent of all other attributes [12]. This assumption is considered naïve [13]. Despite this assumption, its accuracy is still comparable to other sophisticated classifiers and has proved effective in many practical applications [14] [12] [15]. The popularity of naïve Bayes classifier has increased and it is adopted by many due to its computational efficiency, simplicity and performance for real-world problems.

Before we look at the example and its implementation, let us discuss about the mathematical foundation behind Bayes Decision theory. If we have \( p_1(x, y) \) as probability of a piece of data belonging to class 1 and \( p_2(x, y) \) as probability of same data belonging to class2. As per Bayes Decision theory to choose the class with higher probability by following two rules:

- \( p_1(x, y) > p_2(x, y) \), then the class is 1
- \( p_1(x, y) < p_2(x, y) \), then the class is 2

To explain in brief the probabilities \( p_1 \) & \( p_2 \) are the conditional probabilities \( p(x|y) \) and the mathematical formula for the conditional probability is given below:

\[
P(B|A) = \frac{P(A \text{ and } B)}{P(A)}
\]
Bayes’ rule is used to manipulate the conditional probabilities which gives the mathematical calculation behind swapping the symbols inside conditional probability as follows:

\[
P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}
\]

Combining the conditional probabilities and Bayes’ rule from the above equations we can rewrite the Bayesian Classification rule:

- \( p(c_1 | x, y) > p(c_2 | x, y) \), then the class is 1
- \( p(c_1 | x, y) < p(c_2 | x, y) \), then the class is 2

But we don’t have the values for \( p(ci | x, y) \) so by applying Bayes’ rule, it is rewritten as

\[
p(c_i | x, y) = \frac{p(x,y|c_i)p(c_i)}{p(x,y)}
\]
8.2. Example with Python Code

To explain Naïve Bayes elaborately we will look at its famous real-life usage: Email Spam filtering. The first scholarly publication on Bayesian spam email filtering was by Sahami et al. [11]. A Naïve Bayes classifier [12] simply uses Bayes’ theorem on the context classification of each mail, with a naïve assumption that the words included in the email are independent to each other. Data which is emails in this case are collected in text files and are provided to the algorithm. This data is prepared by parsing the text into token vectors which is the input for the algorithm. Inspection of tokens are done as an analysis process to check the accuracy of parsing done. We train the algorithm using training data and the testing is done eventually to calculate the error rate over a set of documents. We would build a complete program which would classify a group of documents and print misclassified ones on the screen with the error rate.
from numpy import *

def createVocabList(dataSet):
    vocabSet = set([])
    for document in dataSet:
        vocabSet = vocabSet | set(document)
    return list(vocabSet)

def trainNB0(trainMatrix, trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
    p0Num = ones(numWords); p1Num = ones(numWords)
    p0Denom = 2.0; p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vec = log(p1Num/p1Denom)
    p0Vec = log(p0Num/p0Denom)
    return p0Vec, p1Vec, pAbusive
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0

def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec

def textParse(bigString):
    import re
    listOfTokens = re.split(r'\W*', bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]
def spamTest():
    docList= []; classList = []; fullText = []
    for i in range(1,26):
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)
    trainingSet = range(50); testSet=[]
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet:
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    for docIndex in testSet:
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
        print "classification error",docList[docIndex]
    print 'the error rate is: ',float(errorCount)/len(testSet)

print spamTest()
8.3. Explanation

The first process in this example would be to convert the words in the documents into a vector with numbers. We would start with `createVocabList()` function which will create an empty set (python data type) and append the set with new set from each document using | operator for producing union of two sets. Set holds only the unique words from the documents. Next as a continuation let us see about `bagOfWords2VecMN()` function which takes the vocabulary list and the input document and gives an output of vector with numbers representing the frequency at which the word in vocabulary is present at the given document.

Since the words has been converted to numbers let us now calculate the probabilities with these numbers to predict the class: Spam or Ham. To perform this, we would call `trainNB0()` function which works in the following stepwise manner:

- Count the number of documents in each class
- For each training document and for each class if a word occurs increment the count for that word
- For each class and for each word divide the word count by the total number of words to get conditional probabilities
- Return the conditional probability for each class

Next `classifyNB()` function is used to perform Bayesian Decision theory in which the class with higher conditional probability becomes the predicted output class. In the function, we would perform element-wise multiplication between two vectors and add up
all the values for the words in vocabulary which is added to logarithmic value of class. 1 is returned if probability of class1 is greater than class2 or 0 for vice versa.

Now the next function *textParse*() would parse the text file into a list of strings which eliminates any word under two characters long and converts them into lowercase. The last function *spamTest*() automated the spam classifier and follows the following steps:

- Initiate vectors, load and parse text files from spam and ham folder into those vectors
- 10% of input data is randomly chosen as test vector and remaining as training set, the probabilities will be computed only form the training set
- When a test set is selected, the algorithm will remove them from training set called as cross validation
- The last loop iterates through all the items in test data and creates word vectors from them and vocabulary using *bagOfWords2VecMN()* function
- Call for *trainNB*() function to calculate the probabilities needed and iterate through each test set and classify each email in the test dataset
- When the email is not classified correctly error count is increased and the final error percentage along with the word list of the email which got misclassified is printed
8.4. Results

We can infer from Figure 11 that due to our code which chooses test and training dataset randomly the output shows different error rates at each instance also showing the email document which got misclassified. Hence the Naïve Bayes algorithm for spam testing has been successfully implemented.

Figure 11. Output Screen of Naïve Bayes Algorithm showing different error rates
9. Conclusion

This study is not a comparison between three classification algorithms because each algorithm has been given different machine learning problems to tackle which fits their strength accordingly. But I would say this paper as a brief study about three important and mostly used classification algorithms: k-Nearest Neighbors, Decision tree and Naïve Bayes. Each example scenarios have been implemented with Python code using an open source data science tool called Anaconda powered by Python 2.0 with inbuilt scientific modules such as NumPy and Matplotlib. The results have been clearly given after each implementation along with the detailed explanation of the python code.
10. Acknowledgement

I would like to thank Professor Bo Yuan for his mentoring and guidance to complete this thesis on Classification Algorithms from Machine Learning. This report was composed for EE I9900 – Master’s Thesis – Spring 2017 semester and presented at City College of New York.

I also thank my family and friends for their continuous encouragement and more support without them this would not be possible.
11. References


