Mr

Mark Mburu

Submitted for the Degree of Master of Science in

Your MSc Programme

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

June 16, 2014

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 10000

Student Name:

Date of Submission:

Signature:

Abstract

This project is an audio feature extraction and classification with the ECS-10 data set and ESC-50 dataset audio dataset .ECS-10 audio data is included. It consists of 10 classes of different environmental sounds (person sneezing, person coughing, etc.). The main objective of this project is to compare classification accuracies for the 6 tested classifiers. The dependencies we are using for this project are Librosa (audio loading, audio visualization and feature extraction), Sci-kit learn, Keras (Theano backend), NumPy, Matplotlib and Pandas (data visualization). The scripts for feature extraction and classification have been added as ipynb files and are all loaded in the Google Colab. Running feature_extraction.py creates a NumPy array for features (feature.npy) and one for labels (label.npy). These files will be saved in the content directory. The audio extracted from the dataset are MFCC, Chroma, Mel spectrogram, Tonal centroid feature and Spectral contrast. The classifiers implemented are:

- Recurrent Neural Network (RNN)
- Support Vector Machine (SVM)
- Naive Bayes (NB)
- Random Forest (RF)
- Multilayer Perceptron (MLP)
- K Nearest neighbors (KNN)
- Convolutional Neural Network (CNN)

Direct comparison between classifiers can't be done yet since their parameters haven't been tuned to optimize accuracy yet. Out of 400 audio samples, the test set consisted on the 33% of this.

- SVM: 81.7%
- RF: 83.33%
- NB: 68.94%
- KNN:61.88%
- CNN: 78.125% (100 epochs)
- MLP: 79.3% (100 epochs)
- RNN: 72.00% (100 epochs)

To improve the accuracy obtained we can compute other features: MFCC + ZCR features improve classification accuracy for speech, noise and music labels to see if it also works for the 10 classes, tune optimization hyper parameters (for every classifier): Weight initialization, decaying learning rate and data scaling and feature normalization (MFCC).

Contents
Table of Contents

1. Introduction

In the recent decades machine learning has gained a lot of traction and has grown not just become a buzz word but something key in our day to day lives. More and more organizations and people are adopting machine learning as part of their operations. Classification, which in its basic of forms is the categorization of various chunks of data into different categories, leverages the advancements in machine learning. Now, classification tasks that previously required human involvement can be done by machines which have proved to be not only faster but also more accurate.

The challenge comes in finding the suitable classification model to use in a given problem. The choice of classification model is very vital when it comes to the success or failure of a given classification task. With different models having different performances based on the data that is being processed it is vital to find the perfect model that is most suitable for the dataset that is to be analyzed. Various classification models have different degrees of accuracy based on a number of factors which include the choice of parameters to use, the nature of the dataset, the distribution of the data etc. As humans always try to better the attempts of the previous strides in all walks of life so do, they in the realm of machine learning. Constant attempts are being made to improve current classification models that are available and where possible come up with new and better models that have high accuracy when it comes to classification.

In this project, we look at various classification-based models that are available and implement them using the provided data. The aim of implementing the different classification models is to compare the different performance of the various classification models with regard to the ESC dataset. The dataset comprises of coughs and sneezes in the form of raw recordings. Due to the fact that non-speech sound recognition has not received major attention despite its importance in dictating and being an indication of a person's health, the data provided a suitable yet important data to analyze.
In this project various models are created using this data so as to access accuracy score obtained from the different conformal predictions. The performance of the different models is recorded and comparisons are made between the different models.

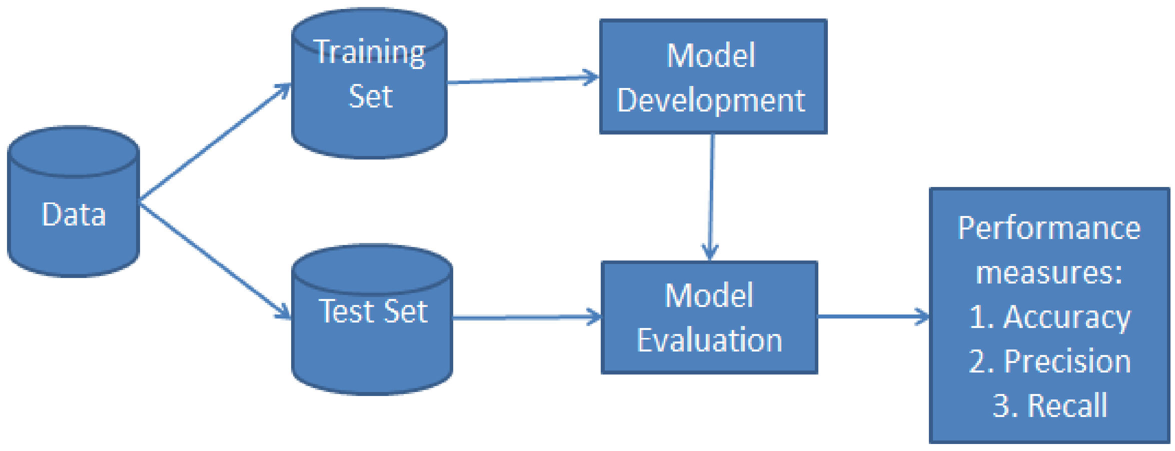
The different classification-based conformal predictors that this project look at include the following: Support Vector Machine, K Nearest Neighbors, Recurrent Neural Network, Naive Bayes Random Forest, Multilayer Perceptron and Convolution Neural Network. These conformal predictors are implemented in order to detect various coughing and sneezing sounds from the provided dataset. The goal is to compare the efficiency and accuracy of the different classification-based conformal predictors and draw a conclusion as t which among them is most efficient on the dataset.

2. Background Research

2.1 Naïve Bayes

This is a simple algorithm that serves to classify large amounts of datasets. It can be utilized in a diverse range of applications for the purpose of classification, for instance in span detection where it can be used to classify spam emails , text classification where is used to classify product ratings and comments from consumers, sentiment analysis where its used to categorize opinions from pieces of texts, and various recommendation systems. Naive Bayes relies on the Bayes theorem and thus leverages probabilities in order to make predictions.[1]

The starting point to any classification problem is first gaining understanding to the problem at hand and followed by the identification of possible features that may be used, it also includes the identification of labels. Here, features refer to independent variables which have a given impact on the various labels which are the dependent variables. [2]For instance, when you consider a case of loans offered by a bank, the manager might have to access various information regarding the customer. This information may include the age, loan track record, occupation, the credit score etc. These characteristics that the manager can use are referred to as features and they can be applied on a given model in order to enable the model make classification and categorize various customers. There are two phases associated with classification, the first phase which is made up of the leaning and the second phase made up of the evaluation of the model.



1 'Naive Bayes Classifiers - Geeksforgeeks' (*GeeksforGeeks*, 2020) <https://www.geeksforgeeks.org/naive-bayes-classifiers/> accessed 23 July 2020.

2 'Sklearn.Naive_Bayes.Gaussiannb — Scikit-Learn 0.23.1 Documentation' (*Scikit-learn.org*, 2020) <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html> accessed 23 July 2020.

The model's training is by using a given dataset known as the training set in the learning phase and the classifier performance is tested in the evaluation phase. The evaluation of performance is on the basis of diverse parameters which may include the accuracy of prediction, errors in prediction,[3] precision of results or positive predicted data which is the fraction of relevance among the retrieved instances, and recall which is also referred to sensitivity is the fraction of the total amount of relevant instances that were.

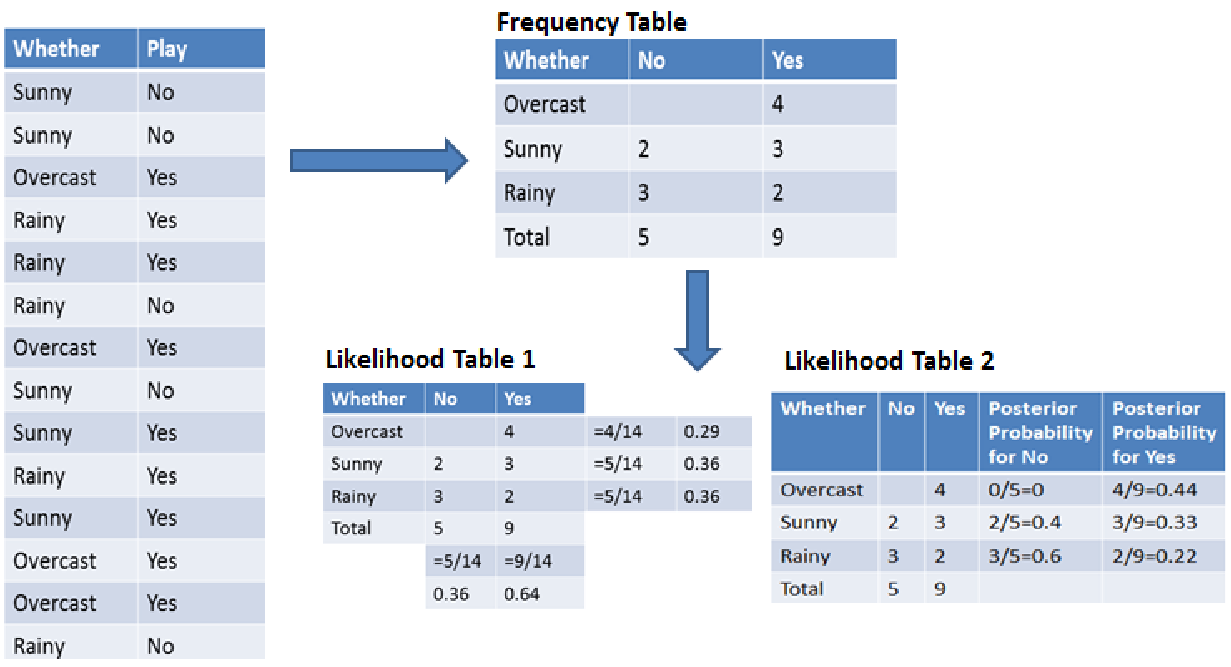
The NB classifier is foundation-ed on a technique that is in Bayes Theorem and it is known as statistical classification. It falls under the supervised machine learning algorithms and it can be termed as accurate, efficient and fast algorithm. They are very fast when classifying large datasets and they work on the assumption that the effect derived from any feature in a particular class is considered independent with regard to other independent variables. A customer who has applied for a loan doesn't depend on his/her history transactions, other loans and income, location, and his/her age. We consider these features independent even if they are interdependent. It is considered as naïve because it assumes that features are interdependent which simplifies computation and this is one of the reasons its fast on large dataset. The class conditional independence is the assumption that features are interdependent.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- Regarded as the probability prior of h is the probabilistic hypothesis P(h) that regardless of any give data h remain true.
- P(D): Independent of the hypothesis, this is referred to: prior probability.
- P(h|D): When given the data D we get the probability hypothesis h which is P(h|D)
- P(D|h): Regarded to as the posterior probability. This represents the probability, when given hypothesis h is true, of data d.

3 (2020) <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c> accessed 23 July 2020.[4]

Naive Bayes follows the following steps to calculate the probability of a particular event. We will first calculate the prior probability of the various class labels then we will find the likelihood probability that is associated with the individual attributes that is found in each class. The values are then fed to the formula and the posterior probability is calculated. Lastly, we shall identify the class with the highest probability when assigned inputs that belong to the class with higher probability.



Whether	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	Yes
Rainy	Yes
Rainy	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table

Whether	No	Yes
Overcast		4
Sunny	2	3
Rainy	3	2
Total	5	9

Likelihood Table 1

Whether	No	Yes		
Overcast		4	=4/14	0.29
Sunny	2	3	=5/14	0.36
Rainy	3	2	=5/14	0.36
Total	5	9		
	=5/14	=9/14		
	0.36	0.64		

Likelihood Table 2

Whether	No	Yes	Posterior Probability for No	Posterior Probability for Yes
Overcast		4	0/5=0	4/9=0.44
Sunny	2	3	2/5=0.4	3/9=0.33
Rainy	3	2	3/5=0.6	2/9=0.22
Total	5	9		

We use the frequency of these two tables and the likelihood tables after simplifying posterior and prior probability calculations.[4] These two tables above help in calculating the posterior and prior probability. The occurrences of labels that belong to all features are contained in frequency table. The likelihood of the tables is the likelihood in the second that shows the posterior probability and the likely hood in the second tables that shows prior probabilities of labels. When calculating the probability of playing if the weather was overcast.

2.1.1 Probability of playing:

prob (overcast or yes) prob(yes) / prob (overcast) *is equal to prob (yes or overcast)* ....(first)

- The prior probabilities calculations are:

46 R and Sunil Ray, 'Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples' (*Analytics Vidhya*, 2020) <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/> accessed 23 July 2020.

    - prob(overcast): 0.29
    - prob(yes): 0.64
- Th posterior probabilities calculation is:
    - prob (overcast or yes): 0.44
- Putting the posterior and prior probabilities in the first equation (first)
    - Prob (yes or overcast): 0.98

2.1.2 The Probability of not playing:

Prob (overcast or no) Prob(no) / Prob (overcast) *is equal to Prob (no or overcast)* ....(second)

- The prior probabilities calculations are:

  Prob(overcast): 0.29

  Prob(no): 0.36
- The posterior probabilities calculations are:

  Prob (overcast or no): 0
- Putting the posterior and prior probabilities in the second equation

  Prob (No or Overcast): 0

Clearly the 'Yes' class has a higher probability and we can conclude that whenever there is an overcast weather, no sport is played.

Take a situation where you want to evaluate the probability a sport being played in the event that it is overcast as well as mild temperature.

2.1.3 The Probability of playing:

Prob (temp is mild or weather is overcast or play is yes) *Prob (Play is Yes)* is equal to Prob (temp is mild or weather is overcast or play is yes) ....*(first)*

Prob (Overcast or Yes) is equal to *Prob (Mild or Yes)* Prob (Weather is Overcast, Temp is Mild or Play is Yes) ………. *(second)*

- The prior probabilities calculation is: Prob (Yes) = 0.64
- The posterior probabilities calculations are: Prob (overcast or yes) = 0.44 Prob (Mild or Yes) = 0.44

- Putting the Posterior in the second equation (second) Prob (Weather is Overcast, Temp is Mild or Play is Yes) is equal to 0.1936
- Putting the posterior and prior in the first equation (first) Prob (Play is Yes or Weather is Overcast, Temp is Mild) is equal to0.124

2.1.4 The Probability of not playing:

Prob (Weather is Overcast or play is No, Temperature is Mild) is equal to Prob (Weather is Overcast, Temperature is Mild or Play is No) *Prob (Play is No) ....(third)*

Prob (Weather is Overcast, Temperature is Mild or Play is No) is equal to Prob (Weather is Overcast or Play is No) *P (Temp is Mild* or Play is No) ………. (fourth)

- To calculate the prior: Prob (No)= 0.36
- To calculate the posterior: Prob (Weather is Overcast or Play is No) is equal to 0/9 = 0 P (Temperature is Mild or Play is No) =0.4
- Putting the posterior probabilities in the fourth equation (fourth) Prob (Weather is Overcast, Temp is Mild or Play is No) = 0
- Putting the posterior and prior probabilities in the third equation (third) Prob (Play is No or Weather is Overcast, Temp is Mild) =0

The 'Yes' class has a higher probability, so it can be said that when there is an overcast weather player still manage to play the sport.

2.1.5 Advantages of Using Naïve Bayes

- It provides prompt and accurate predictions on top of its simplicity
- The cost of computation is very low
- Handles large datasets effectively
- More suitable to discrete variables as compared to continuous
- It is possible to apply it with problems involving more than one class
- Its performance in analytic problems such as text is very good

- Performs superior to other models such as logistic regression, provided independence of features

2.1.6 Disadvantages of Naive Bayes

- In ppractice, it is very rare for features to be completely independent.
- A posterior probability of zero arises in the situation where a training tuple is not available which makes the model incapable of any prediction. This challenge is referred to as Zero Probability/Frequency.

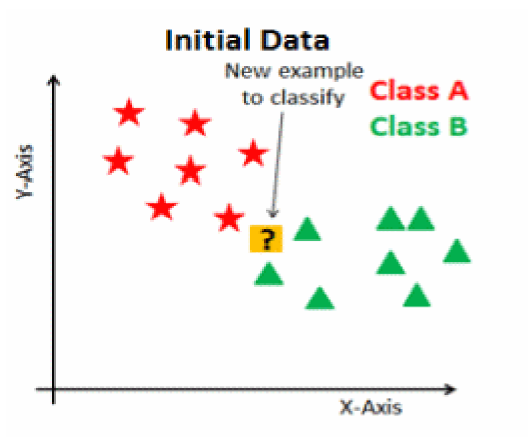
2.2 KNN Classification Algorithm

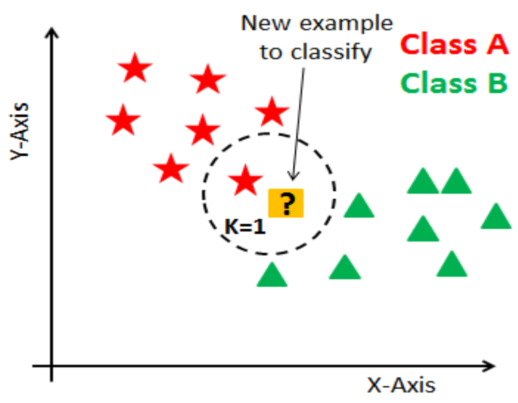
The applications of this algorithm cut across various disciplines ranging from political science, financial predictions, image, speech and video recognition, the detection of handwriting etc. Financial institutions may leverage this algorithm in order to make prediction regarding their clients' credit rating. Banks access the risk associated with a particular client who wants a loan. Political scientists may group voters into those who might and might not vote. All these are just but a few applications of the KNN algorithm.[5]

The KNN algorithm is a distribution independent classification algorithm that picks its model structure from the provided data. This comes in handy in practical use since most real-world problems and data do not conform to the ideal assumptions that are put in mathematical classes. Its model generation does not depend on any training data points thus it is regarded to as a lazy algorithm. The training data is utilized during the testing, resulting in a fast training and a slowed down testing characterized with high memory and time requirements. At its worst, KNN requires large chunks of time to allow for the scanning of the provided data points and in turn additional memory is required in order to store the scanned data.

The 'k' in KNN represents the number of neighbors that a classification model checks in order to come up with a proper classification. More often than not it is provided as an odd number and in the situation where k=1, the algorithm is hen referred to as the nearest neighbor algorithm. For instance, a point P1can be predicted by first finding the closest point to P1 and assigning the label of the point to P1.

[5] 'K-Nearest Neighbor(KNN) Algorithm For Machine Learning - Javatpoint' (*www.javatpoint.com*, 2020) <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning> accessed 23 July 2020.

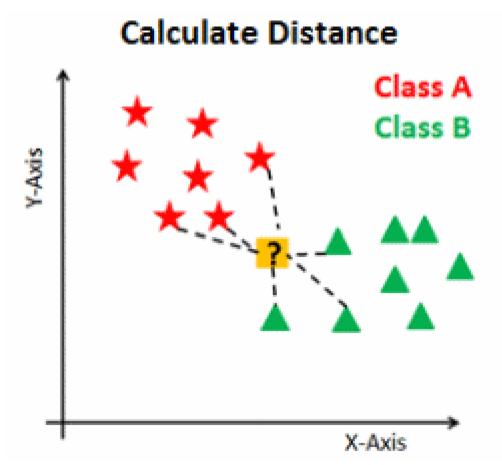




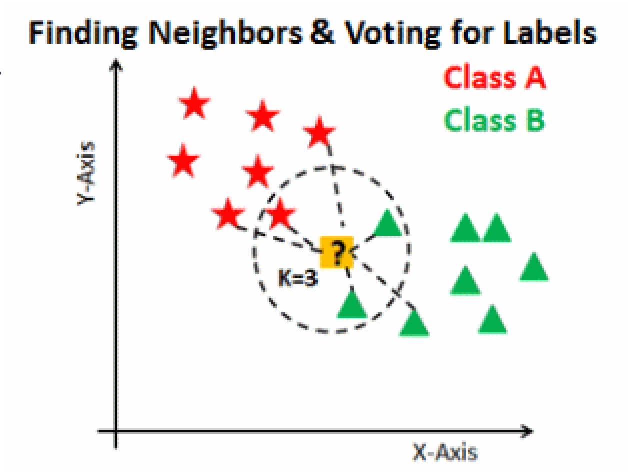
Suppose we have a point P1 which has to be predicted. The first action to be taken would be to first find the 'K' (where K is an integer) closest point(s) to P1 and look at the majority vote. The class that will have the highest vote among those K closest to P1 will be taken as the predicted class. In order to get the shortest distance between points the Euclidean distance is used. Other distances such as the Hamming, the Manhattan and the Minkowski may also be used.

In order to make predictions, KNN follows the following steps:

- Calculation of distances between points
- Establishing of the neighbors closest to a point
- Voting for a particular label



Eager



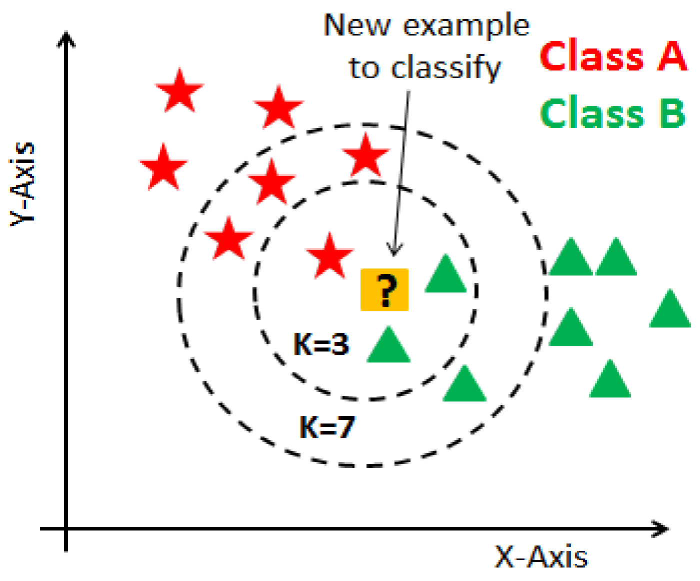
learners are learners that are ready for prompt classification, active in classifying and eager to classify data points that have not been classified. When provided with the training points it first constructs a generalized model then proceeds to predict new points. Lazy learning classifies the data points at the last minute as well as the model does not need to train or learn and it uses all the data points during the prediction time. For Lazy learner the training dataset is merely stored and waits until the time classification has to be done. It performs if the dataset has a tuple. It uses similarities based on different tuples in order to perform generalization and classify. [6]This type of learners does most of their work in the testing phase compared to the training phase unlike in eager training and learning methods. The training points of various instances are stored and learning is as a result of the stored instances, hence they are known as instance-based learners. The problem of over fitting is caused by the Increase in dimensionality. The data needed will be characterized with exponential growth as the number of dimension increase in order to avoid over fitting. Curse of Dimensionality is the challenge where there are a lot of dimensions. To handle this issue, principal component analysis has to be conducted before the application of any machine learning algorithm, or the feature selection approach can be used. Other measures such as cosine similarity, less affected by the issues of high dimensions, can be preferred as research indicates that the Euclidean distance is not particularly useful where the dimension is large. Now that we understand the working mechanism of KNN, we have established a suitable way of choosing the optimal number of neighbors and understand the effects of it to the classifier. The choice of neighbors (K) in KNN is an important attribute that has to be selected during the model building process and it acts as a variable that controls the prediction model.

There is no set optimal number of k neighbors that will be suitable for every situation, meaning that depending on the dataset unique parameters have to be selected for optimal results.[7]
Where a small number for K is used, noise usually has a higher impact on the results and on the flip side, when a large number is used for K it results in high computational requirements.
Using a large number for K results in a smooth decision boundary which implies that the variance will be low but the bias will be high. However, using a lower value for K results in more flexible fits characterized by high variance but low bias.

Where the number of classes is even, an odd number is usually used for the value of K. By generating various replications of the models while applying different values for K, the performance of the different variations can be assessed.

6 'Machine Learning Basics With The K-Nearest Neighbors Algorithm' (*Medium*, 2020) <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761> accessed 23 July 2020.

7 Introduction R) and Tavish Srivastava, 'K Nearest Neighbor | KNN Algorithm | KNN In Python & R' (*Analytics Vidhya*, 2020) <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/> accessed 23 July 2020.



2.2.1 Advantages of K-Nearest Neighbor (KNN)

Compared to other classification algorithms KNN is faster in the training phase. It is very simple and it is also instance-based thus the generalization model does not need to be trained which is very useful in case of nonlinear data. The average of the K closest neighbors value computes the output value for the object and can be used to solve regression problem,

2.2.2 Disadvantages of K-Nearest Neighbor (KNN)

In terms of time and memory consumption, especially in the testing phase, it is costlier. In order to store and predict the entire training dataset the algorithm requires large memory. It makes use of the Euclidean distance in order to find the nearest neighbors and to scale the data. Euclidean distance is affected by magnitudes. It’s not suitable for large dataset and the features with low magnitude weigh less than those with high magnitude

2.2.3 How to improve KNN

It is highly recommended to normalize data on the same scale for better result. The normalization range is considered to be between 0 and 1. For large dimensional data, KNN is not suitable. In cases of high dimensions, to enhance the performance we need to reduce the dimension. Fixing missing and invalid values in the dataset helps us in improving results.

2.3 Random Forest

Various decision trees are combined in order to come up with a random forest. For instance, a good number of individual trees operating as a single ensemble belong to a random forest. Our model predictions are made up of the class with the most votes and a prediction is then split to get those of individual trees in a random forest. The fundamental concept that supports the random forest is the know-how of crowd. The random forests are able to work so well as a result of the high number of uncorrelated trees that operate as a committee thus, they outperform models made up of individual constituents. The intricacy behind is that the correlation between the models has to be low. In the case of investments that have low correlation e.g. stock and bonds, this is how they come together to form a portfolio that is usually more valuable than the individual parts. It is clear that models that are uncorrelated are able to produce more accurate predictions compared to the individual predictions. As long as tress do not have constant all error in the same direction, they insulate each other against errors that may affect them individually. While many other trees will be right, some trees may be wrong, but like a group they are capable of moving towards the right direction as a result of the trees that are correct. In order for random forest to perform well there has to be an actual signal that is present in the feature thus allow for models that are built to depend on the feature to perform better than the case in random guessing. The predictions that the individual trees make should have little correlation.

The concept of having many uncorrelated models is critical hence showing an example to demonstrate it in order to sink in is fitting. In the case where you were playing the game, to produce a number I made use of uniform random forest. In order to win the game, I have to generate a number that is >=40, which makes your victory chance 60% thus I will have to give you money. Now I offer you some choices to follow. Either you can play 100 times where you will bet $1 for each bet or you can play 10 time but for this you can bet $10 for each bet or you could play just once and place a bet of $100. Note that for each of the games the expected values are similar which are (0.60*1 + 0.40*-1) *100 = 20 or (0.60*10 + 0.40*-10) *10 = 20 or 0.60*100 + 0.40*-100 = 20. Using the Monte Carlo simulation, it is possible to visualize the results. Running 10,000 simulations for the different game options i.e. the 100 plays in the first choice will be simulated 10,000 times.

The[8] distributions that are obtained are very different from each other ranging from positive and[9] characteristically narrow to binary even though the expected values are the same**.** The first choice where the game is played 100 times provides the best opportunity to make money since 97% of the simulations make money. For the second choice where we play 10 times, we get a drastic decline as a result of the % for making money dropping to 63% of the carried out of the simulations, which if looked from the side of losing money is a high chance. The third option offered a chance to play the game once. This had a further reduction in the percentage of simulations to 60%. Thus, the outcome of the distributions is completely different although they share a similar expected value. In order to make the most amount of money it will be better to split the $100 into different bets so as to maximize the chances. This is so because the different plays will be independent of each other. If we look at the individual trees as different plays that we had, the random forest is the same. Thus, the chances of obtaining a correct prediction will always increase as the number of uncorrelated trees in our random forest model, which follows a similar trend to how it went in our chances of making money in the game. The random forest uses two methods to ensure that the individual trees are uncorrelated with those of the other trees in the model. The methods are discussed below:

2.3.1 Bagging or Bootstrap Aggregation

Bagging occurs where minute alterations to the nature of the training set sometimes cause an outcome that is significantly different tree structures because the sensitivity of decision trees to the data that was used during training is very high. So, to get result in different trees Random forest enables the individual trees to sample from the dataset randomly with replacement while taking advantage of the sensitivity of decision trees to the data that was used to train them in a process known as bagging. In bagging the training set is not sub settled into small chunks that are used to train individual trees. On the contrary, given a sample that a size S, each tree will still have a training set that is of size S. The difference is that a random training set is obtained from the data with replacement. For instance one tree may be allocated the following instance [1, 2, 2, 3, 6, 6] if our

8 'Chapter 5: Random Forest Classifier' (*Medium*, 2020) <https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1> accessed 23 July 2020.[6]

9 '3.2.4.3.1. Sklearn.Ensemble.Randomforestclassifier — Scikit-Learn 0.23.1 Documentation' (*Scikit-learn.org*, 2020) <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> accessed 23 July 2020.[1]

training data was [1, 2, 3, 4, 5, 6]. Note, we sample with replacement and that is why the length of both of the lists is six and “2” as well as “6” appear more than once in the training set that was randomly selected.

2.3.2 Feature Randomness

Feature Randomness is when splitting the node in a normal decision tree, the different features that are possible are considered and a choice is mode for the feature that results in the biggest separation of the observation in the right and the left if the node. In a random forest individual trees can only pick from the pool containing the subset of features present.[10] This results in an increase in diversification among the trees and a reduced correlation as a result of increased variation among the trees in the models. For random forest when building different sets of models that are uncorrelated, the models usually have an expected return that is positive, thus when put together there is a massive alpha earned in a portfolio. The random forest makes use of feature randomness and bagging during the construction of individual trees in order to obtain a forest of trees that is uncorrelated thus capable of making better predictions as compared to the prediction that are made by individual trees on their own. The features that are used need to have some degree of predictive power associated to them in order for the decision tree to come up with accurate prediction. The trees in the forest need to be uncorrelated. The algorithm on its own always attempts to provide some degree of correlational among the trees using feature randomness, our choice of features and parameters also has an impact on the ultimate correctness.

2.3.3 Merits of Random Forest

- It is a very accurate and functionally robust as a result of the large number of trees that are involved.
- They are not affected by the problem of over fitting as a result of its nature of taking the averages of the individual predictions.
- It is suitable for classification as well as regression tasks

10 'Random Forests Classifiers In Python' (*DataCamp Community*, 2020) <https://www.datacamp.com/community/tutorials/random-forests-classifier-python> accessed 23 July 2020.

- They are robust in handling missing values by replacing the missing values with median values and the use of proximity-weighted averages.

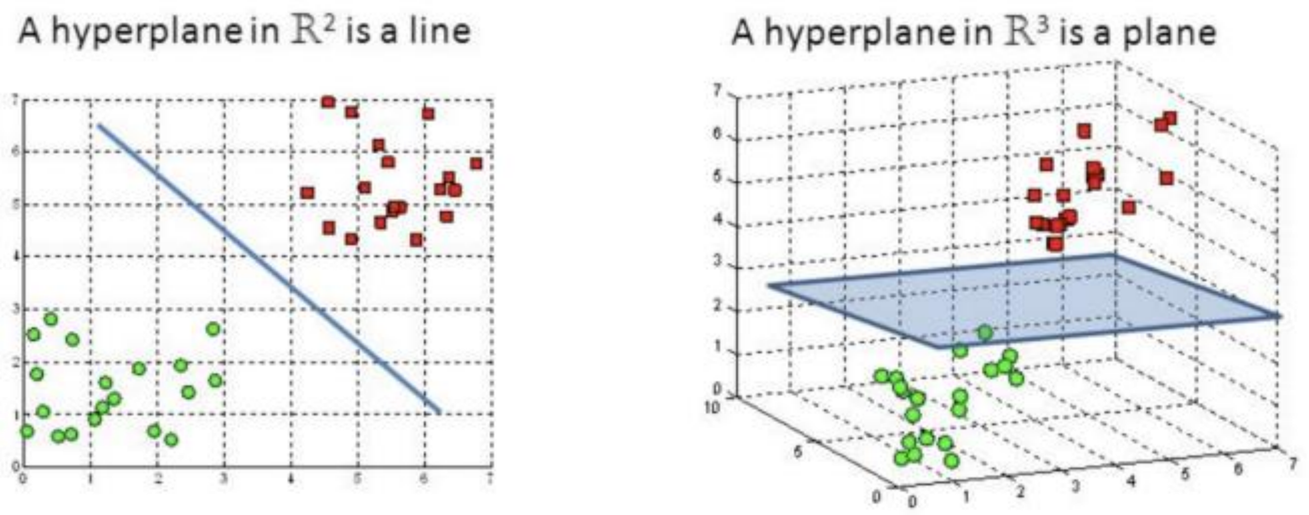
2.3.4 Demerits of Random forest Classifier

- Because of it having multiple decision trees predictions are done at a slow speed. For a final prediction to be made all the individual trees that are present have to first make a prediction using the data then voting is done.

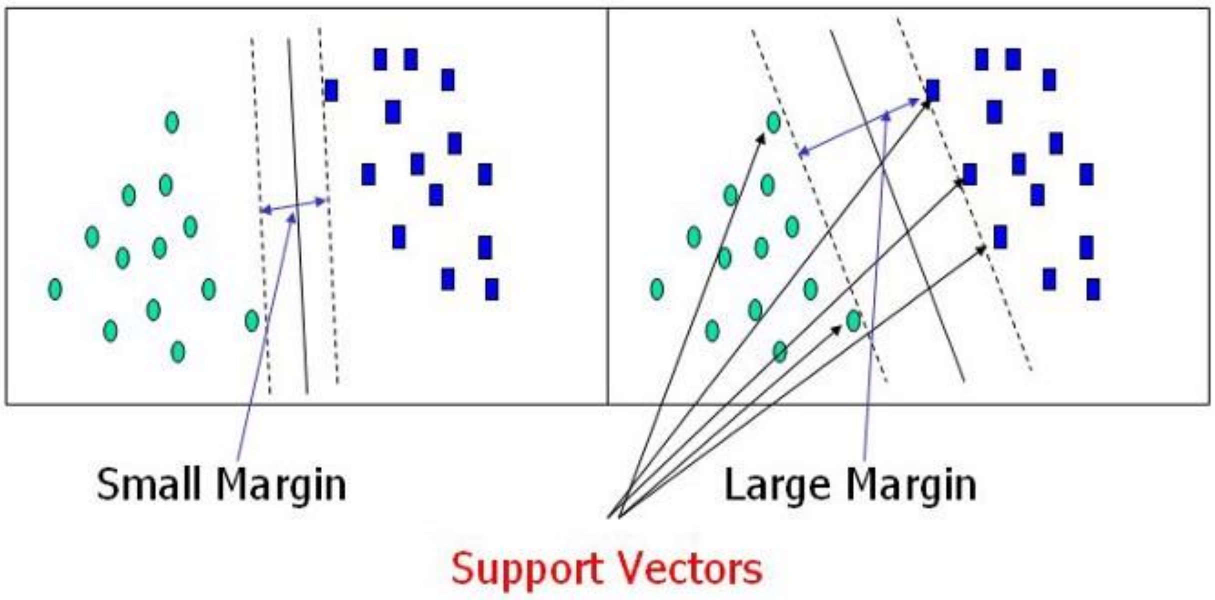
- The interpretation of a random forest model is quite difficult compared to that of a simple decision tree, where a decision is arrived by following the path to the tree

2.4 Support Vector Machine

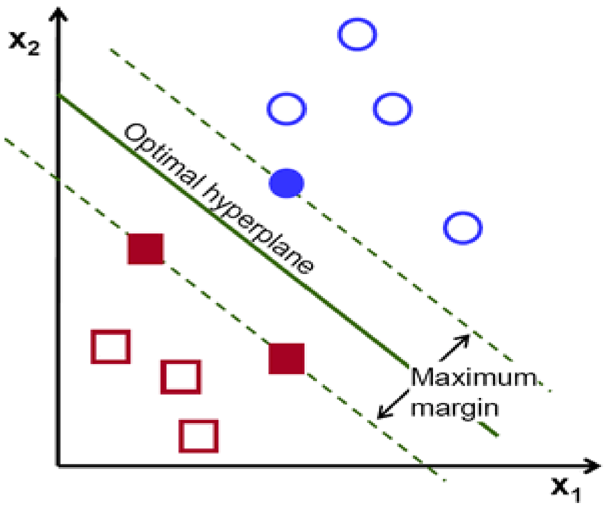
This technique aims at maximizing the margins by finding decision boundaries which separate the various classes. Margins are those dots closet to the line and the perpendicular distances between the lines. Support Vector Machine also aims at finding the most suitable line give the constraint that the correct classification of any class by merely looking at the hyperplanes since the hyperplanes are capable of classifying different classes correctly and maximizes margins to conduct optimization. We will use the concept of margin and separate hyperplane. In the case of a 2D space, hyperplane is considered to be an n minus 1-D subspace and a hyperplane will always tend to be 1D, just a simple line. Hyperplane will be 2-dimension, for a 3-dimension space which results in a plane slicing the cube.



Assuming the label y is either positive (1) or negative (-1) and all those three lines below is known as separating hyperplanes. Separate hyperplane constraint is written mathematically above because they all share the same properties. In a perfect case a linear separable case, the constraint can be achieved by Support Vector Machine.



But if the case of non-separable, we will need to loosen it. Margin occurs where a plane that I usually a line says X, and the distance (perpendicular) is calculated from X to the forty dots, resulting in



various distances where the shortest becomes the margin. The margin then is the distance from either side of the of the line that is dashed. It can be considered as the stretching that is the widest between the blue and the green dots. Linear separable cases of support vector machine are constrained where the individual cases attempt to make observations from the proper direction regarding the hyperplane and then select the line that is the most suitable in order to maximize the margin, i.e. the Euclidean distance from the dots to the hyperplane. Being constrained that classification of classes has to be correct, they attempt to get the hyperplane that maximizes the margins. However, in reality, the datasets are never separable linearly, thus the 100% classification made by hyperplanes is never met. Also support vector machines introduces two concepts in order to handle non-linearly separable instances. The two concepts are **Kernel Tricks** and **Soft Margin**. Upon addition of a blue dot on the green dotted cluster, it results in a situation where the data in not separable linearly. This is a problem and its solutions are **Soft Margin** that will tolerate miss-classified dots by first finding a line that separates them and is robust to one or two dots; the other solution is Kernel Tricks that attempt to establish a decision boundary that is non-linear. Support Vector Machine is capable of tolerating two miss-classifications, first is where dots may be correct side facing the margin but are on facing the wrong side regarding to the decision boundary or a situation where the dot is facing the wrong side of both the particular decision boundary and the margin. When Soft Margin is applied, a couple of dots may be tolerated by SVM, as it attempts to strike a balance between margin maximization and maximizing the miss-classification. The degree of tolerance is the amount of tolerance that we are willing to give when we are trying to find the decision boundary and is usually a key parameter considered in the SVM. Its representation is the penalty term "C" and it depends on the support vector less and less as the margin narrows in Sklearn. The “C” value increase with an increase in the number of penalties that the SVM gets during miss-classification. Kernel Tricks make use of existing features, in some cases it may apply some transformations and in other cases it might develop new features. The newly created features help in finding the nonlinear decision boundary and are key to SVM. The s*vm.svc ()* method in Sklearn has a number of available choices such as rbf and sigmoid that are usually callable just like the kernel/transformation. The polynomial and the RBF are the two most widely used kernels. The Polynomial kernel serves as a processor whose function is to generate new features. This is done through the process of combining the various existing features. i.e. for instance, a feature that existed say X = np. array ([-2, -1,0, 1,2]) and a label like Y = np.array([1,1,0,1,1]), finding a line the clearly separates the two will be impossible. However, the application and use of a transformation X^2can come in handy as it leads to a new feature that is X = np.array([4,1,0, 1,4]). Now it becomes possible to draw a line that will separate the dots. By applying various polynomial features, the SVM with a polynomial kernel is capable of generating boundaries that are non-linear. The Radial Basis Function (RBF) measures the distance from as particular dot to all the other dots and generates new features. The **gamma (γ)** and Φ **(x, center)** control the impact of the newly generated features. The value of the gamma increases with an increase in the impact of the features to the decision boundary. As a result of combining the strengths of soft margin and that of kernel tricks, SVM is capable of structuring the decision boundary for cases they are linearly non-separable. The wiggling of the SVM decision boundary is controlled by gamma and hyper-parameters like when there is more penalty given to SVM when it miss-classified the higher the C, resulting in reduced wiggling observed on the decision boundary. When there are more influences on the feature data points, it leads to an increase in gamma implying that more wiggling of the boundary.

2.5 Multilayer Perceptron

It is composed of a couple of neurons combined in linear layers. Starting with the input layer whereas the last layer is the output and sandwiched between the two is the hidden layer for a simple three-layered network multiplayer Perceptron[11]. We take the output from the output layer and the

11 'Multilayer Perceptron — Deeplearning 0.1 Documentation' (*Deeplearning.net*, 2020) <http://deeplearning.net/tutorial/mlp.html> accessed 23 July 2020.

input is fed at the input layer. To increase the complexity of the model based on our task, the number hidden layers sandwiched can be increases to satisfaction. The aim of a Feed Forward Network is to aggregate a given function f (), which is the most prevalent model of the neural network. The Multilayer Perceptron attempts to obtain the best approximation of a given classifier (say **y=f*(x)**) which maps a given input X into the corresponding Y output by using a map y = f (x; θ) using the most suitable learning parameter θ. The Multilayer Perceptron is made up of chained functions that result to f(x) = f (3) (f (2) (f (1)(x))), a network with three layers or functions. Each of these functions or layers are composed of units that perform a particular affine transformation when provided a given linear aggregation of inputs and thus y = f (WxT + b) is a formula that represents each layer. In this formula W represents the weights that are in the layers, f denoted the activation function, b is the bias vector and x represents the input that in some cases it can be the output of the layer before the[12] current. The layers or functions of a Multilayer Perceptron are made up of interconnected full layers since every individual unit in the layers is joined to the units of the preceding layer. The nature of the parameters is such that they are independent across the different layers which means that the different units have unique weights associated with them. The various inputs are linked with labels which defines the particular class or in the case of supervised learning, is provided with the data. When given a particular input, a prediction is given as an output. A loss function is used to measure and access the performance of the classifier. The loss s high in the situation where the predicted class is wrong. At the time of training the model problem of over fitting and under fitting occurs sometimes. In the situation that this happens the performance of the model on test data is very good but poorly on test data. An optimization procedure is required so as to train the network and for this we need an optimizer and loss function. This process minimizes the loss function by getting the values of different weights. To get a lower loss we use a popular strategy that refines the weights iteratively and initializes them. The gradient of the loss function defines the movement direction. The learning rate which defines the moving amount is necessary in each iteration.

The various input-output connections are usually described by an activation function. This improves the model's flexibility when it comes to situations that are arbitrary. Sigmoid and Relu are some of the most common activation functions. In order to train the model first is the forward pass, then the

12 Jason Brownlee, 'Crash Course On Multi-Layer Perceptron Neural Networks' (*Machine Learning Mastery*, 2020) <https://machinelearningmastery.com/neural-networks-crash-course/> accessed 23 July 2020.

calculation of the loss and finally the backward pass. During the forward pass of the model, weights are multiplied by the provided inputs and the bias is added at the different layers. The predicted output obtained from the model can be used to calculate the loss and the real output has the label Expect out. Back propagation is used to calculate the loss, and based on various outputs obtained, after which the weights are adjusted through the use of gradient.

13

13 'A Beginner's Guide To Multilayer Perceptrons (MLP)' (*Pathmind*, 2020) <https://pathmind.com/wiki/multilayer-perceptron> accessed 23 July 2020.[5]

[14]2.6 Convolution Neural Network

This algorithm assigns important weight values to provided inputs and also give various biases to the different objects that are found in the input data thus differentiating one from another is simple. When we compare it to other classification algorithms it has a lower pre-processing. CNN through training is capable of learning characteristics unlike in hard engineered previous methods.[15] The connectivity of CNN is synonymous to that of the brain of humans. Response to stimuli is restricted to a particular field in the neuron, which is covered by overlapping fields. The spatial dependencies as well as the temporal dependencies are capable of being caught by a CNN. The dataset fitting can be performed better when the number of parameters and weights are reduced. By ensuring that the critical features are not lost, CNN reduces a dataset into a processing friendly form. When we are trying to design an architecture, it is important to reduce the given dataset so as to make the feature learning process easier and more scalable.

2.7 Recurrent Neural Network.

Closely resembling the human brain, this is an algorithm designed to spot and detect patterns that may be present in data. The fact that they resemble the complex human brain merits the name. They rely on an advance perception of machines, the clear labeling of data and clustering of input that is usually raw in order to correctly interpret sensory data. All the data in the real world that ranges from various sounds, captured images etc. all must be transformed into a form that is comprehensible by the neural network.

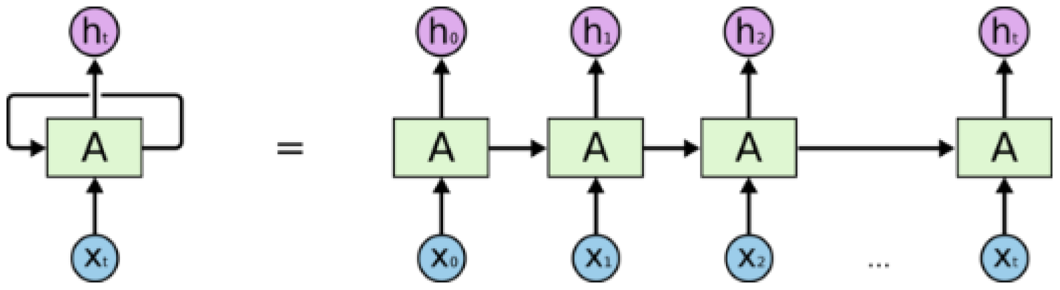
RNN is characterized by an internal memory functionality and can be considered to be a generalization of a feed-forward based neural network. Where no cycle is formed by the connection among the units.

Artificial NN are usually comprised up of a significant number of very intricately connected elements referred to as neurons that work in unison in order to achieve a common task. They are effectively applied to solve situations where the provided dataset is not sequential nor is it dependent on a particular time constraint, a case common in supervised leaning models. Outputs of processing and prediction is dependent on the previous processes since this network is recurrent. A decision is arrived at after the analysis of the provided input in collaboration with the output obtained from the previous data that was inputted. The obtained result is duplicated and sent to the network. They have internal state memory which they can use to process sequences of inputs data unlike the feed-forward

14 'A Comprehensive Guide To Convolutional Neural Networks—The ELI5 Way' (*Medium*, 2020) <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> accessed 23 July 2020.[3]

15 'Cs231n Convolutional Neural Networks For Visual Recognition' (*Cs231n.github.io*, 2020) <https://cs231n.github.io/convolutional-networks/> accessed 23 July 2020.

neural networks. RNN are extensively applied in the recognition of handwriting and speech recognition which are usually unsegmented. Unlike the design of other neural networks, in RNN there is relationships between the different inputs. The figure below depicts an illustration of an RNN.



An unrolled recurrent neural network.

Its outputs h (0) when combined with X (1), input after it takes the X (0) from the sequence of input. After this first step h (0) and X (1) is the input that is passed to the next step. The context is always stored while training because h (1) leads on to the input X (2) that proceeds to the next step. Its current state formula is:
h(t) is current hidden state and is the current input x(t) and is a function that characterizes the previous state h(t-1). The symbol theta acts as the operational parameters of the function f.
When we apply the activation function we get:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

The figure below shows how unfolding maps from left to right in which the black square represents a particular interaction that occurs with some delay in time (i.e. 1), the change in state from t to that state at t+1. Opening and the sharing of various parameters at different positions: thus, resulting in a reduced number of parameters that have to be estimated.

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

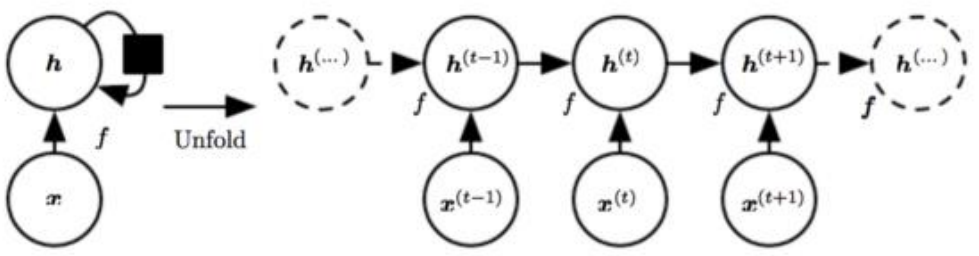


Figure 10.2: A recurrent network with no outputs. This recurrent network just processes

2.7.1 Merits of Recurrent Neural Network:

- Recurrent neural network supports for the modeling of sequences of data allowing each sample to depend on the past data
- It is possible to extend the functionality of RNN with the use of the convolutional layers.

2.7.2 Demerits of Recurrent Neural Network:

- Characterized with vanishing of the gradient and various exploding problems
- It is very difficult to train an RNN
- Where it applies Relu as its activation function, it's unable to process long sequences.

3. Experiment

3.1 Aim

The goal is to compare the efficiency and accuracy of the different classification-based conformal predictors and draw a conclusion as to which among them is most efficient on the dataset.

3.2 Preparatory Work

Data Preparation:

In order to implement the classification algorithms so as to make predictions it is necessary to perform raw data cleaning and transformation in order to process and analyze it.[16] Preparation is the term that refers to the process of ensuring that raw data is clean and then transformed to an analysis friendly nature. This process is a vital process in relation to data analysis and it involves the actual formatting of data, handling missing and incorrect inputs and finally putting various data together to get a richer data to use.[17]
The code below illustrates how I transformed the dataset to get the features and labels as NumPy arrays file that I later used to train and predict on my models.

```
import glob
import os
import librosa
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import specgram
import soundfile as sf

def feature_extraction(file_name):
    X, sample_rate = librosa.load(file_name)
    if X.ndim > 1:
            X = X[:, 0]
            X = X.T

# Get features
stft = np.abs(librosa.stft(X))
mfccs = np.mean(librosa.feature.mfcc (y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
# 40 values
# zcr = np.mean(librosa.feature.zero_crossing_rate)
chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T, axis=0)
```

16 Jason Brownlee, 'What Is Data Preparation In A Machine Learning Project' (*Machine Learning Mastery*, 2020) <https://machinelearningmastery.com/what-is-data-preparation-in-machine-learning/> accessed 24 July 2020.

17 Data Preparation, 'Data Preparation For Machine Learning | Datarobot Artificial Intelligence Wiki' (*DataRobot*, 2020) <https://www.datarobot.com/wiki/data-preparation/#:~:text=What%20is%20Data%20Preparation%20for,uncover%20insights%20or%20make%20predictions.&text=Improperly%20formatted%20%2F%20structured%20data.> accessed 24 July 2020.

```
contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,
axis=0)
tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
sr=sample_rate).T, # tonal centroid features
                              axis=0)

# Return computed features
return mfccs, chroma, mel, contrast, tonnetz

# Process audio files: Return arrays with features and labels
def parse_audio_files(parent_dir, sub_dirs, file_ext='*.ogg'):  ## .ogg audio format
features, labels = np.empty((0, 193)), np.empty(0)  # 193 features total. This can
vary

for label, sub_dir in enumerate(sub_dirs):  ##Enumerate() function adds a counter to
an iterable.
                 for file_name in glob.glob(os.path.join(parent_dir, sub_dir, file_ext)):
           ##parent is audio-data, sub_dirs are audio classes
try:
mfccs, chroma, mel, contrast, tonnetz = feature_extraction(file_name)
 except Exception as e:
 print("[Error] there was an error in feature extraction. %s" % (e))
continue

extracted_features = np.hstack(
[mfccs, chroma, mel, contrast, tonnetz])  # Stack arrays in sequence horizontally
(column wise)
features = np.vstack([features, extracted_features])  # Stack arrays insequence
vertically (row wise).
labels = np.append(labels, label)
print("Extracted features from %s, done" % (sub_dir))
return np.array(features), np.array(labels, dtype=np.int)


# Read sub-directories (audio classes)
audio_directories = os.listdir("audio-data/")
audio_directories.sort()


# Function call to get labels and features
# This sabes a feat.npy and label.npy numpy-files in the current directory
features, labels = parse_audio_files('audio-data', audio_directories)
np.save('feat.npy', features)
np.save('label.npy', labels)
```

The librosa.load() functions takes in the file and returns a sample rate and a 2d array which has two axis the first and the second axis. The first axis represents amplitude samples that are recorded and the second axis represent the number of channels of audio. Then we check the dimension of x is greater than one if its greater than one we slice all the rows and keep the first column(0) else we get the transpose of X in X.T(which means transpose of X).stft basically means short time Fourier transform and we get this by passing X into librosa.stft() and we get the absolute value by passing the result into np.abs() method.mfccs means mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features which concisely describe the overall shape of my data. So, I get the mean value of the values produced by librosa. feature.mfcc() and store it to mfccs variable. Chroma is a vector and it is typically a 12-element feature vector indicating how much energy of each pitch class. To get this I got the mean of the values produced by librosa.feature.chroma_stft(S=stft, sr=sample_rate) method and then transposed the mean to get the vector and stored it in chroma variable. We convert the frequencies into mel scale then we get the mean and transpose it. The tonnetz variable gets the Tonal centroid features for each frame and the we compute the mean and transpose it. Then we return the computed features of mfccs, chroma, mel, contrast and tonnetz. Then we process audio files and return arrays with features and labels. I had checked the audio files and I had found out that they were .ogg format that's why I specified it and then I got the 193 features and the number can vary. Then we iterated over the sub directories and we use the enumerate method that adds counter to an iterable. Then we read the folder and all its sub directories and we use glob to get the path and the we extract the features and store them in the variables that we had defined i.e. mfccs, chroma, mel, contrast and tonnetz. We use try except in just in case of an error occurring during this process. Then we get the Stack arrays in sequence horizontally (column wise) and Stack arrays in sequence vertically (row wise) that we store in extracted_features and features respectively. Then we append the labels and we return an array of features. Then we read the sub directories and I had renamed my Es-10 Folder to audio-data. Then we sort the sub directories and call the parse_audio_files() to get the labels and features and I used np.save() to save the feat.npy file. and label.npy file.

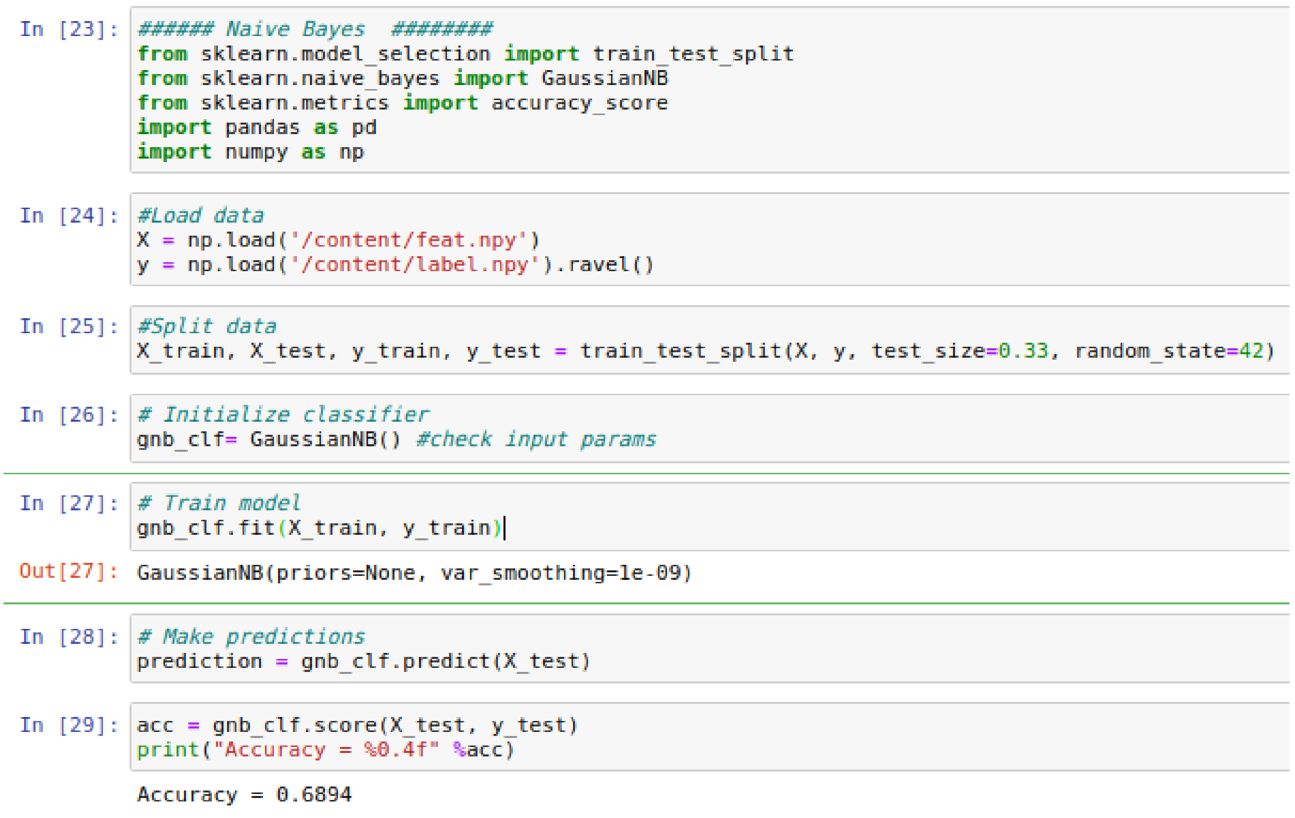
The code above shows how I transformed my dataset which was a folder containing other folders with audio files. I first import the necessary modules that I needed which were glob which helped in finding the path names of the dataset, of which is a module that provides of using dependent functionality, librosa which I used for audio analysis since the dataset had audio files, sound file which I used to read audio samples, numpy which we use to work with arrays and matplotlib which we use to visualize our dataset. After importing the dataset, I defined a function feature extraction that took one parameter of file name and used librosa to load the audio files that were passed as filenames from our dataset. After loading the file names, I got the features and computed them and returned the computed features that were ready to be processed. Then I processed the features to return an array of features and labelled and stored them in feat.npy file and label.npy file which represented the features and labels respectively

3.3 Method

3.3.1 Building Classifier in Scikit-learn Naive Bayes Classifier

To use scikit-learn knn classifier algorithm, I first imported the modules that I needed to use with my dataset which are:

- Numpy to read my data
- Sklearn's accuracy score to calculate the accuracy of my model
- GaussianNB Classifier from sklearn naïve Bayes.
- Train test split to split my data



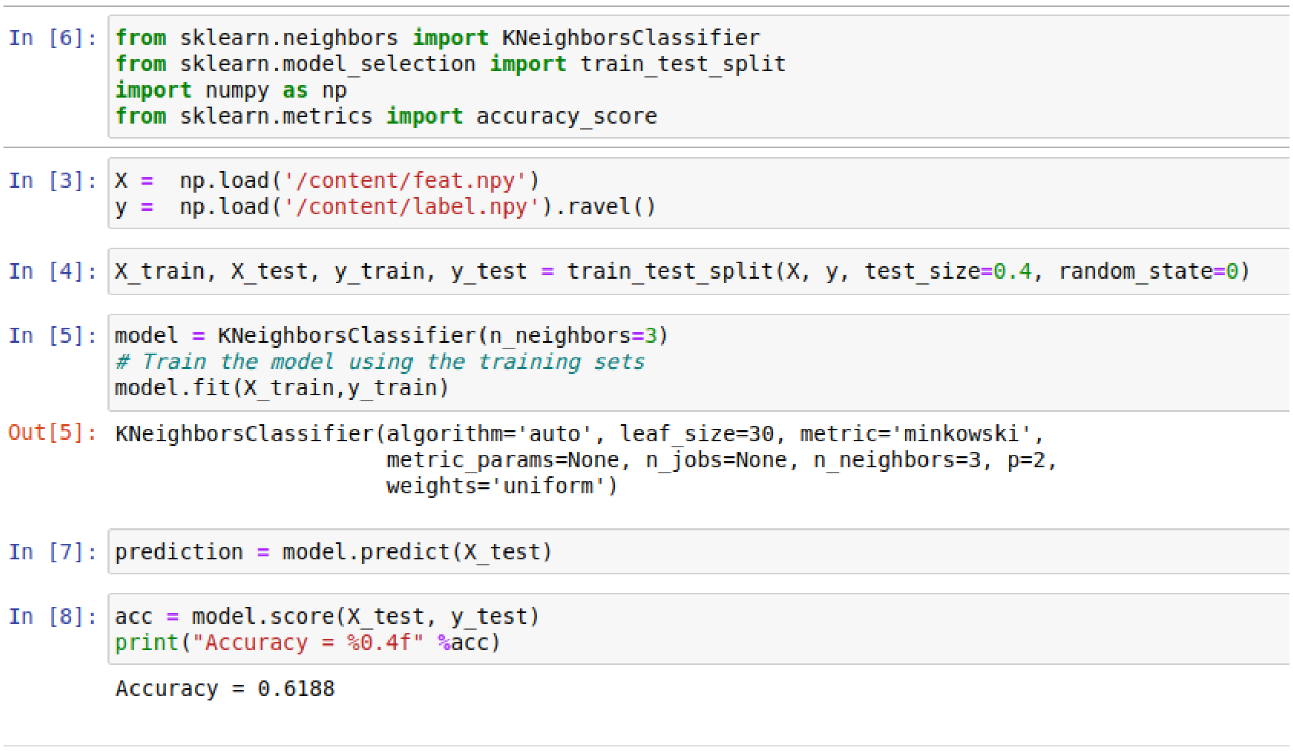
```
In [23]: ###### Naive Bayes  ########
         from sklearn.model_selection import train_test_split
         from sklearn.naive_bayes import GaussianNB
         from sklearn.metrics import accuracy_score
         import pandas as pd
         import numpy as np

In [24]: #Load data
         X = np.load('/content/feat.npy')
         y = np.load('/content/label.npy').ravel()

In [25]: #Split data
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

In [26]: # Initialize classifier
         gnb_clf= GaussianNB() #check input params

In [27]: # Train model
         gnb_clf.fit(X_train, y_train)

Out[27]: GaussianNB(priors=None, var_smoothing=1e-09)

In [28]: # Make predictions
         prediction = gnb_clf.predict(X_test)

In [29]: acc = gnb_clf.score(X_test, y_test)
         print("Accuracy = %0.4f" %acc)

         Accuracy = 0.6894
```

The above code is how I implemented the Naïve Bayes classifier on the dataset. First, I imported the necessary modules then I loaded the data and stored them and X and y variables in the case where the x variable stored the features whereas the y variable stored the labels. I then divided the dataset using the splitting method provided by sklearn to obtain two sets of data, the training as well as the testing set. Then I initialized the classifier which is GaussianNB and I assigned it to gnb_clf variable. Then I trained my data by using the fit method and I passed two parameters or arguments into it which are the training set. Then I did my predictions by passing the testing set into the predict method of gnb_clf variable. Then I printed out the prediction and I performed calculations for the accuracy of my model which was 0.6894 or 68.94%.

3.3.3 Building Classifier in Scikit-learn KNN Classifier

To use scikit-learn knn classifier algorithm, I first imported the modules that I needed to use with my dataset which are:

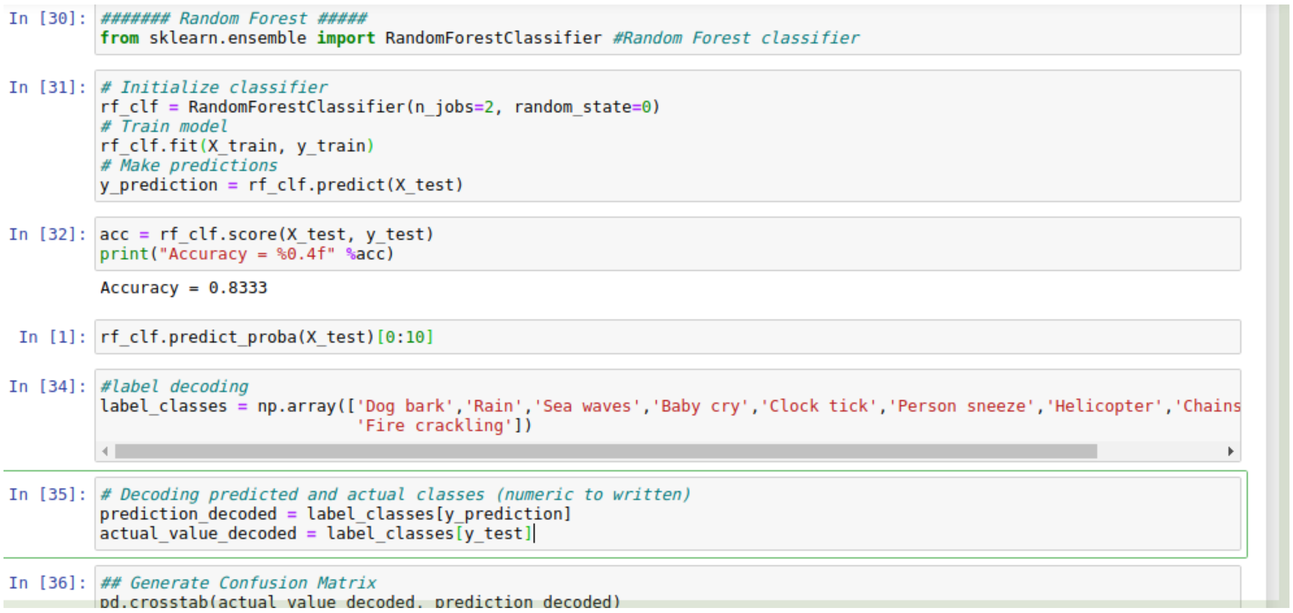
- Numpy to read my data
- Accuracy score from sklearn to calculate the accuracy of my model
- K Neighbours Classifier from sklearn neighbours which is the algorithm
- Train test split to split my data



```
In [6]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import train_test_split
        import numpy as np
        from sklearn.metrics import accuracy_score

In [3]: X =  np.load('/content/feat.npy')
        y =  np.load('/content/label.npy').ravel()

In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)

In [5]: model = KNeighborsClassifier(n_neighbors=3)
        # Train the model using the training sets
        model.fit(X_train,y_train)

Out[5]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                             weights='uniform')

In [7]: prediction = model.predict(X_test)

In [8]: acc = model.score(X_test, y_test)
        print("Accuracy = %0.4f" %acc)

        Accuracy = 0.6188
```

The above code is how I implemented the KNN classifier on the data. First I imported the necessary modules then I loaded the data and stored them and X and y variables where x stored the various features or the independent variables whereas y stored the labels or the dependent variable .I then proceeded to split the dataset into training set and testing set using train test split method from sklearn . Then I initialized the classifier which is K Neighbors Classifier with three neighbors and I assigned it to model variable. Then I trained my data by using the fit method and I passed two parameters or arguments into it which are the training set. Then I did my predictions by passing the testing set into the predict method of model variable. Then I printed out the prediction and I performed calculations for the accuracy which was 0.6188 or 61.88%.

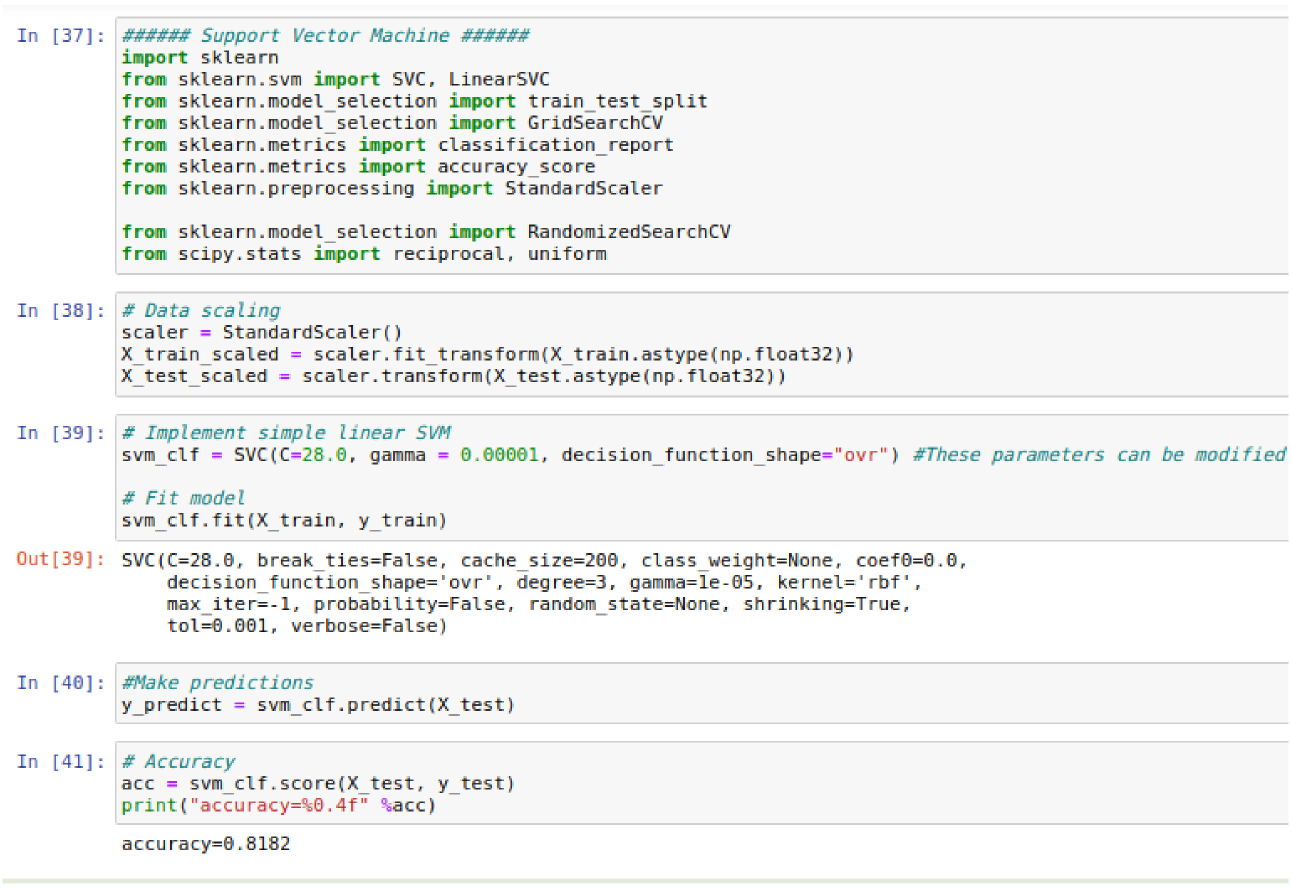
3.3.4 Building Classifier in Scikit-learn Random Forest



```
In [30]: ####### Random Forest #####
         from sklearn.ensemble import RandomForestClassifier #Random Forest classifier

In [31]: # Initialize classifier
         rf_clf = RandomForestClassifier(n_jobs=2, random_state=0)
         # Train model
         rf_clf.fit(X_train, y_train)
         # Make predictions
         y_prediction = rf_clf.predict(X_test)

In [32]: acc = rf_clf.score(X_test, y_test)
         print("Accuracy = %0.4f" %acc)

         Accuracy = 0.8333

 In [1]: rf_clf.predict_proba(X_test)[0:10]

In [34]: #label decoding
         label_classes = np.array(['Dog bark','Rain','Sea waves','Baby cry','Clock tick','Person sneeze','Helicopter','Chains
                                   'Fire crackling'])

In [35]: # Decoding predicted and actual classes (numeric to written)
         prediction_decoded = label_classes[y_prediction]
         actual_value_decoded = label_classes[y_test]

In [36]: ## Generate Confusion Matrix
         pd.crosstab(actual_value_decoded, prediction_decoded)
```

The above code is how I implemented the random forest classifier on the dataset. First, I imported the necessary modules then I loaded the data and stored them and X and y variables where x stored the various features or the independent variables whereas y stored the labels or the dependent variable. I then proceeded to split the dataset into training set and testing set using train test split method from sklearn. Then I initialized the classifier which is Random forest classifier with two arguments of random state and jobs and I assigned it to rf_clf variable. Then I trained my data by using the fit method and I passed two parameters or arguments into it which are the training set. Then I did my predictions by passing the testing set into the predict method of rf_clf variable. Then I printed out the prediction and I calculated the accuracy of my model which was 0.8333 or 83.33%. This was the highest accuracy I got from all classifiers that I implemented.

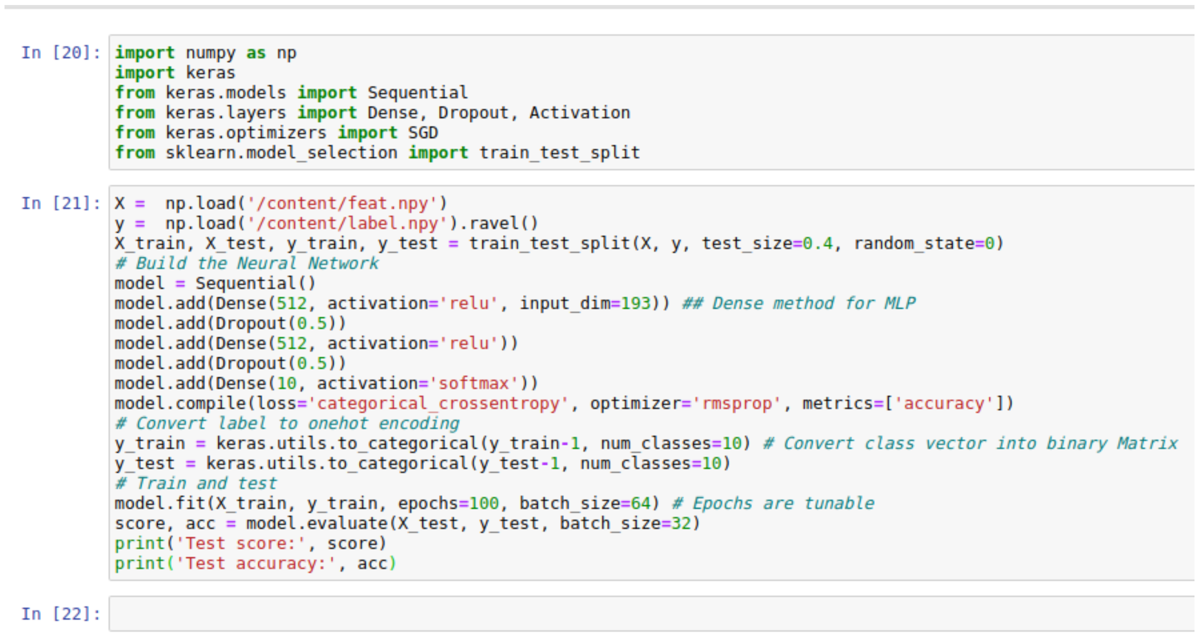
3.3.5 Building Classifier in Scikit-learn Support Vector Machine



```
In [37]: ###### Support Vector Machine ######
         import sklearn
         from sklearn.svm import SVC, LinearSVC
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import classification_report
         from sklearn.metrics import accuracy_score
         from sklearn.preprocessing import StandardScaler

         from sklearn.model_selection import RandomizedSearchCV
         from scipy.stats import reciprocal, uniform

In [38]: # Data scaling
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train.astype(np.float32))
         X_test_scaled = scaler.transform(X_test.astype(np.float32))

In [39]: # Implement simple linear SVM
         svm_clf = SVC(C=28.0, gamma = 0.00001, decision_function_shape="ovr") #These parameters can be modified

         # Fit model
         svm_clf.fit(X_train, y_train)

Out[39]: SVC(C=28.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma=1e-05, kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [40]: #Make predictions
         y_predict = svm_clf.predict(X_test)

In [41]: # Accuracy
         acc = svm_clf.score(X_test, y_test)
         print("accuracy=%0.4f" %acc)

         accuracy=0.8182
```

The above code is how I implemented the support vector machine classifier on the dataset. First I imported the necessary modules then I loaded the data and stored them and X and y variables where x stored the features or the independent variables and y stored the labels or the dependent variable .Then I split the data into training set and testing set using train test split method from sklearn. Then I initialized the classifier which is SVC with three parameters of c which I set to 28,gamma which I set to 0.00001 and decision function shape that I set to over and these parameters can be modified to improve the accuracy of the algorithm and I assigned it to svm_clf variable. Then I trained my data by using the fit method and I passed two parameters or arguments into it which are the training set. Then I did my predictions by passing the testing set into the predict method of svm_clf variable. Then I printed out the prediction and I calculated the accuracy of my model which was 0.8194 or 81.94%.

3.3.6 Building Classifier in Scikit-learn Multilayer Perceptron

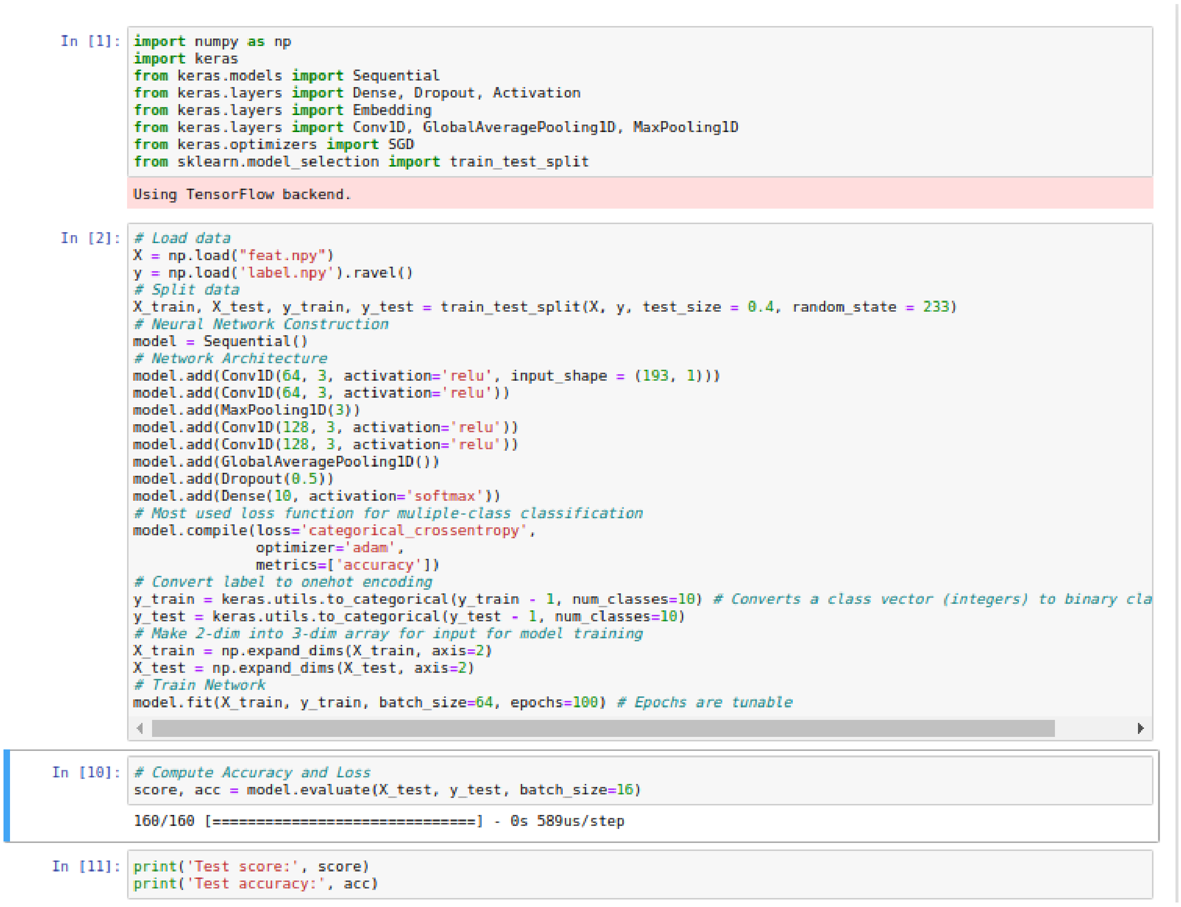


```
In [20]: import numpy as np
         import keras
         from keras.models import Sequential
         from keras.layers import Dense, Dropout, Activation
         from keras.optimizers import SGD
         from sklearn.model_selection import train_test_split

In [21]: X =  np.load('/content/feat.npy')
         y =  np.load('/content/label.npy').ravel()
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
         # Build the Neural Network
         model = Sequential()
         model.add(Dense(512, activation='relu', input_dim=193)) ## Dense method for MLP
         model.add(Dropout(0.5))
         model.add(Dense(512, activation='relu'))
         model.add(Dropout(0.5))
         model.add(Dense(10, activation='softmax'))
         model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
         # Convert label to onehot encoding
         y_train = keras.utils.to_categorical(y_train-1, num_classes=10) # Convert class vector into binary Matrix
         y_test = keras.utils.to_categorical(y_test-1, num_classes=10)
         # Train and test
         model.fit(X_train, y_train, epochs=100, batch_size=64) # Epochs are tunable
         score, acc = model.evaluate(X_test, y_test, batch_size=32)
         print('Test score:', score)
         print('Test accuracy:', acc)

In [22]:
```

```
Test score: 3.84693864583969l3
Test accuracy: 0.793749988079071
```

The code above shows how I implemented the multilayer perceptron to train and predict the outcome of my dataset. First, I started by importing the modules I needed to use and I was using keras to implement my neural network. Then I loaded the data into X and y which represented the features as well as labels respectively. Then I split the dataset into training and testing set using train test split method from sklearn. Then I started building my Neural Network model and I started by initializing Sequential and assigning it to the model variable. Then I added several layers of my network with different activation functions which are relu and softmax. Then I used the loss function of multiple class classification which was categorical cross entropy and I used adam's optimizer and set the metrics to accuracy. Then I converted the labels into onehot encoding to convert class vectors to binary class matrix for both the training set and testing set. Then I converted the input for model training from a two dimensional to three dimensional and trained the network with 100 epochs until it converged. Then I computed the accuracy and loss of my model and I got test score of 3.8469386458396913 and a test accuracy of 0.793749988079071 which is 79.37499%. This was the highest accuracy of the neural networks I implemented.

3.3.7 Implementing Convolution Neural Network with keras

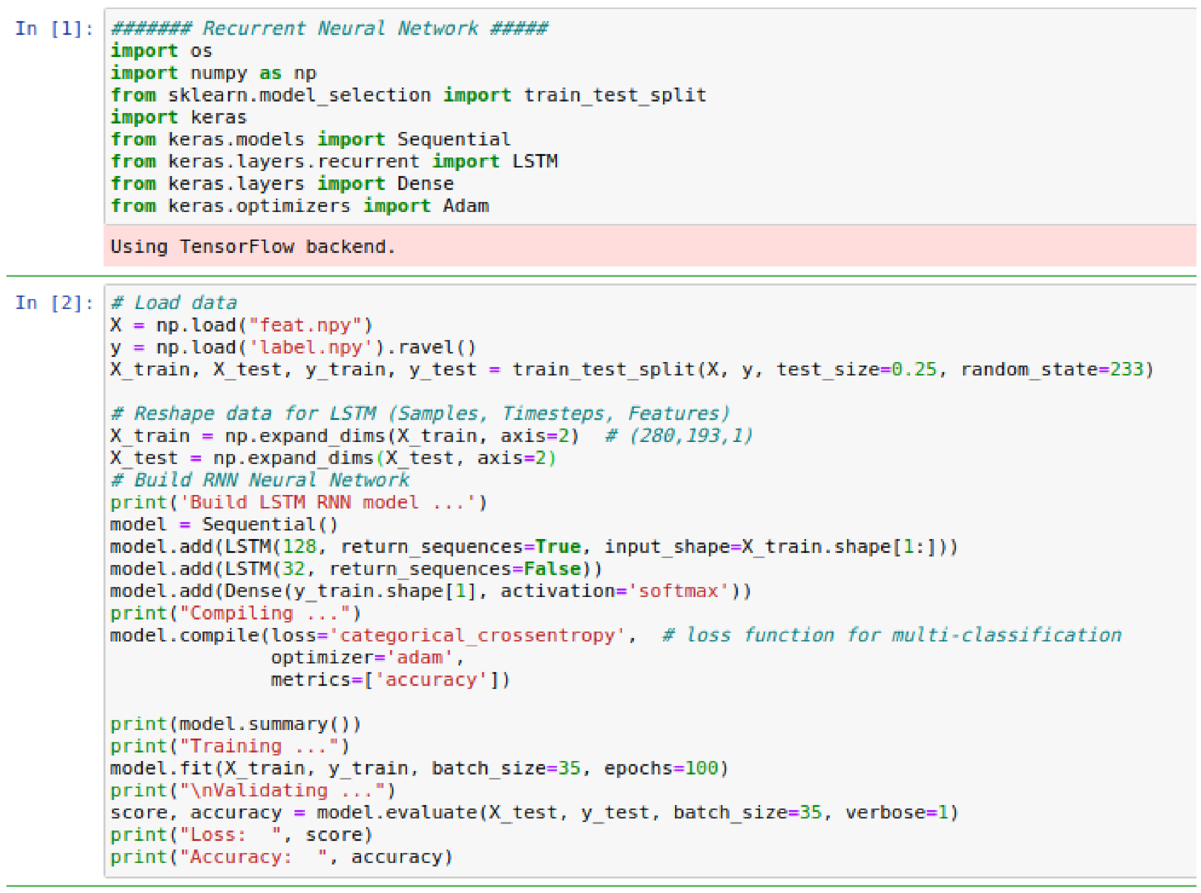


```
In [1]: import numpy as np
        import keras
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation
        from keras.layers import Embedding
        from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
        from keras.optimizers import SGD
        from sklearn.model_selection import train_test_split

        Using TensorFlow backend.

In [2]: # Load data
        X = np.load("feat.npy")
        y = np.load('label.npy').ravel()
        # Split data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state = 233)
        # Neural Network Construction
        model = Sequential()
        # Network Architecture
        model.add(Conv1D(64, 3, activation='relu', input_shape = (193, 1)))
        model.add(Conv1D(64, 3, activation='relu'))
        model.add(MaxPooling1D(3))
        model.add(Conv1D(128, 3, activation='relu'))
        model.add(Conv1D(128, 3, activation='relu'))
        model.add(GlobalAveragePooling1D())
        model.add(Dropout(0.5))
        model.add(Dense(10, activation='softmax'))
        # Most used loss function for muliple-class classification
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
        # Convert label to onehot encoding
        y_train = keras.utils.to_categorical(y_train - 1, num_classes=10) # Converts a class vector (integers) to binary cla
        y_test = keras.utils.to_categorical(y_test - 1, num_classes=10)
        # Make 2-dim into 3-dim array for input for model training
        X_train = np.expand_dims(X_train, axis=2)
        X_test = np.expand_dims(X_test, axis=2)
        # Train Network
        model.fit(X_train, y_train, batch_size=64, epochs=100) # Epochs are tunable

In [10]: # Compute Accuracy and Loss
         score, acc = model.evaluate(X_test, y_test, batch_size=16)

         160/160 [==============================] - 0s 589us/step

In [11]: print('Test score:', score)
         print('Test accuracy:', acc)
```

```
Test score: 0.8873702585697174
Test accuracy: 0.78125
```

The code above shows how I implemented the convolution neural network to train and predict the outcome of my dataset. First, I started by importing the modules I needed to use and I was using keras to implement my neural network. Then I loaded the dataset to X and y which represented the features as well as labels respectively. I proceeded to split the data into training and testing set using train test split method from sklearn. I started building my Neural Network model and I started by initializing Sequential and assigning it to the model variable. Then I added several layers of my network with different activation functions which are relu and softmax and maxpooling. Then I used the loss function of multiple class classification which was categorical cross entropy and I used Adam's optimizer and set the metrics to accuracy. Then I converted the labels into onehot encoding to convert class vectors to binary class matrix for both the training set and testing set. Then I converted the input for model training from a two dimensional to three dimensional and trained the network with 100 epochs until it converged. Then I computed the accuracy and loss of my model and I got test scores of 0.88737 which is 88.737% and a test accuracy of 0.78125 which is 78.125%.

3.3.8 Implementing Recurrent Neural Network with Keras



```
In [1]: ####### Recurrent Neural Network #####
        import os
        import numpy as np
        from sklearn.model_selection import train_test_split
        import keras
        from keras.models import Sequential
        from keras.layers.recurrent import LSTM
        from keras.layers import Dense
        from keras.optimizers import Adam

        Using TensorFlow backend.
```

```
In [2]: # Load data
        X = np.load("feat.npy")
        y = np.load('label.npy').ravel()
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=233)

        # Reshape data for LSTM (Samples, Timesteps, Features)
        X_train = np.expand_dims(X_train, axis=2)  # (280,193,1)
        X_test = np.expand_dims(X_test, axis=2)
        # Build RNN Neural Network
        print('Build LSTM RNN model ...')
        model = Sequential()
        model.add(LSTM(128, return_sequences=True, input_shape=X_train.shape[1:]))
        model.add(LSTM(32, return_sequences=False))
        model.add(Dense(y_train.shape[1], activation='softmax'))
        print("Compiling ...")
        model.compile(loss='categorical_crossentropy',  # loss function for multi-classification
                      optimizer='adam',
                      metrics=['accuracy'])

        print(model.summary())
        print("Training ...")
        model.fit(X_train, y_train, batch_size=35, epochs=100)
        print("\nValidating ...")
        score, accuracy = model.evaluate(X_test, y_test, batch_size=35, verbose=1)
        print("Loss:  ", score)
        print("Accuracy:  ", accuracy)
```

```
Loss:   0.9351997882127762
Accuracy:   0.7200000286102295
```

The code above shows how I implemented the recurrent neural network to train and predict the outcome of my dataset. First, I started by importing the modules I needed to use and I was using keras to implement my neural network. Then I loaded the dataset as X and y which represented the features as well as labels respectively. Then I split the data into training and testing set using train test split method from sklearn. Then I started building my Neural Network model and I started by initializing Sequential and assigning it to the model variable. Then I added several layers of my network with different activation functions which were relu and softmax. Then I used the loss function of multiple class classification which was categorical cross entropy and I used Adam's optimizer and set the metrics to accuracy. Then I converted the labels into onehot encoding to convert class vectors into a given binary class matrix for both the two datasets. Then I converted the input for model training from a two dimensional to three dimensional and trained the network with 100 epochs until it converged. Then I computed the accuracy and loss of my model and I got test score of 0.935199 which is 91.5199% and a test accuracy of 0.7200 which is 72%.

3.4 Conclusion

After running all the algorithm I found out that random forest produced the highest accuracy score among all the supervised learning classification algorithm and its accuracy score was 83% and support vector machine came a second close with an accuracy of 81%.For Neural Networks I used three algorithms and after comparing the accuracy I found out that multilayer Perceptron had the highest accuracy followed by convolution neural network. There is a room to further improve the accuracy of the algorithm and one can try to adjust the hyper parameters and use pipelines which can help improving the accuracy of the supervised learning model.

4. Self-Assessment

While undertaking this work, I was able to get considerable insights that I can channel into my next project. An analysis of my work is given below.

4.1 Strengths

The use of tools and resources: While undertaking this project I had to find the best tools to use for various sections of the project. Knowing the right tool is not enough, as it is necessary to know how to use the tool efficiently as well. I relied on reference tracking tools in order to keep my references well tracked. I used Google Drive to store and access all my project files and be able to work on them remotely.

Writing skills: The skills I have gained throughout my academic life proved to be a boost when it came to writing this report. This skill made the organization of the work to be simple and made the entire writing process a little bit easier.

Organization: By using various project planning tools, it helped me to be organized as I did the work. It also made it easier to track project progress and access the milestones and checkpoints that I was hitting. Being responsive and open to suggestions on how to manage the project also made me more comfortable while undertaking the project.

4.2 Weaknesses

Lack of enough experience with certain areas: During the development process of this project, I had to interact with various tools. Certain tools posed a challenge as they needed me to research on how to best use them in order to archive the goals I had set to hit. The implementation of the CNN with the use of keras was a challenge at first. I have to do some reading on how to implement it properly. The lack of enough experience using certain tools forced me to have to seek clarification and advice from forums and various discussion groups.

Time management: As a result of inefficient project scoping, time became an issue as the project workload towards the end became cumbersome. Hopefully in the next project, having a better project scope and proper planning of how to accomplish the project in the stipulated time will come in handy to help avoid this challenge of time.

4.3 Opportunities

Gaining knowledge on machine learning: This project gave me a chance to dive into the deep sea of machine learning related realm. From the preparation of data to the creation of a model and the fine tuning of a model to obtain the best possible result.
Working with data and data sets through the various phases until arriving to a conclusion regarding a certain issue has made me appreciate the field of data science and especially machine learning.

Advance understanding in python: This project gave me an opportunity to apply various concepts of python language in a practical project. It bolstered my understanding and use of the programming language beyond what I previously had before the start of the project.

5. How to use my project

In order to properly make use of this project which includes the executing of the various scripts the following guide should be followed.

- For the preparation of data, where data has to be transformed into a shape and form that can be utilized by the various models, the audio files are converted into NumPy arrays and the features as well as the labels are saved in a .npy file.
- Run the Jupiter notebook. The Jupyter notebook can be launched from Anaconda navigator. The notebook should open as a tab on your default browser.
- Navigate to the project directory
- Run the knn.ipynb in order to execute the K-Nearest Neighbors Classifier.
- Run the multilayer_perceptron.ipynb in order to execute the Multilayer Perceptron classifier
- Run the RecurrentNeuralNetwork.ipynb in order to execute the Recurrent Neural Network classifier
- Run the convolutionNeuralNetwork.ipynb in order to execute the Convolution Neural Network classifier
- Run Esc.ipynb in order to execute the Naive Bayes, the Random Forest and the SVM classifiers.
- The accuracy of the different classifiers is shown upon executing the various .ipynb files.

6. Appendix

1. convolutionNeuralNetwork.ipynb
2. Esc.ipynb
3. feat.npy
4. feature_extraction.py
5. knn.ipynb
6. label.npy
7. multilayer_perceptron.ipynb
8. RecurrentNeuralNetwork.ipynb

7. References

1. '3.2.4.3.1. Sklearn.Ensemble.Randomforestclassifier — Scikit-Learn 0.23.1 Documentation' (*Scikit-learn.org*, 2020) <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> accessed 23 July 2020
2. 'A Beginner's Guide To Multilayer Perceptrons (MLP)' (*Pathmind*, 2020) <https://pathmind.com/wiki/multilayer-perceptron> accessed 23 July 2020
3. 'A Comprehensive Guide To Convolutional Neural Networks — The ELI5 Way' (*Medium*, 2020) <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> accessed 23 July 2020
4. An Nguyen K, and Luo Z, , *Seventh Workshop on Conformal and Probabilistic Prediction and Applications* (Proceedings of Machine Learning Research 2018)
5. (2020) <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c> accessed 23 July 2020
6. Brownlee J, 'Crash Course On Multi-Layer Perceptron Neural Networks' (*Machine Learning Mastery*, 2020) <https://machinelearningmastery.com/neural-networks-crash-course/> accessed 23 July 2020
7. Brownlee J, 'What Is Data Preparation In A Machine Learning Project' (*Machine Learning Mastery*, 2020) <https://machinelearningmastery.com/what-is-data-preparation-in-machine-learning/> accessed 24 July 2020
8. 'Chapter 5: Random Forest Classifier' (*Medium*, 2020) <https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1> accessed 23 July 2020
9. 'Cs231n Convolutional Neural Networks For Visual Recognition' (*Cs231n.github.io*, 2020) <https://cs231n.github.io/convolutional-networks/> accessed 23 July 2020
10. 'K-Nearest Neighbor(KNN) Algorithm For Machine Learning - Javatpoint' (*www.javatpoint.com*, 2020) <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning> accessed 23 July 2020
11. 'Karolpiczak/Paper-2015-Esc-Dataset' (*GitHub*, 2020) <https://github.com/karolpiczak/paper-2015-esc-dataset> accessed 24 July 2020
12. 'Machine Learning Basics With The K-Nearest Neighbors Algorithm' (*Medium*, 2020) <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761> accessed 23 July 2020
13. 'Multilayer Perceptron — Deeplearning 0.1 Documentation' (*Deeplearning.net*, 2020) <http://deeplearning.net/tutorial/mlp.html> accessed 23 July 2020
14. 'Naive Bayes Classifiers - Geeksforgeeks' (*GeeksforGeeks*, 2020) <https://www.geeksforgeeks.org/naive-bayes-classifiers/> accessed 23 July 2020
15. Preparation D, 'Data Preparation For Machine Learning | Datarobot Artificial Intelligence Wiki' (*DataRobot*, 2020) <https://www.datarobot.com/wiki/data-preparation/#:~:text=What%20is%20Data%20Preparation%20for,uncover%20insights%20or%20make%20predictions.&text=Improperly%20formatted%20%2F%20structured%20data.> accessed 24 July 2020
16. R) I, and Srivastava T, 'K Nearest Neighbor | KNN Algorithm | KNN In Python & R' (*Analytics Vidhya*, 2020) <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/> accessed 23 July 2020
17. R 6, and Ray S, 'Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples' (*Analytics Vidhya*, 2020) <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/> accessed 23 July 2020
18. 'Random Forests Classifiers In Python' (*DataCamp Community*, 2020) <https://www.datacamp.com/community/tutorials/random-forests-classifier-python> accessed 23 July 2020
19. 'Sklearn.Naive_Bayes.Gaussiannb — Scikit-Learn 0.23.1 Documentation' (*Scikit-learn.org*, 2020) <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html> accessed 23 July 2020
20. Zhou X, *Proceedings Of The 23Rd ACM International Conference On Multimedia* (Association for Computing Machinery 2015)