

**GOA UNIVERSITY**

**Taleigao Plateau, Goa-403206**



**PROJECT REPORT ON**

**"FAKE NEWS DETECTION WITH MACHINE LEARNING "**

**submitted in partial fulfillment of the requirement for the award of**

**BACHELOR OF ENGINEERING**

**In**

**Electronics and Telecommunication Engineering**

**Submitted by**

**MELROY BARRETO (P. R. No. -201608294)**

**JAYESH PATEL (P. R. No. -201608375)**

**UTPAL SAHAKARI (P. R. No. -201608399)**

**Under the Guidance of**

**DR. SUPRIYA PATIL**

**Associate Professor, ETC Department**



**Department of Electronics and Telecommunication Engineering**

**Padre Conceicao College of Engineering, GOA-403722**

**2019-20**

**GOA UNIVERSITY**  
**Taleigao Plateau, Goa-403206**



***Certificate***

Certified that the project work entitled "**FAKE NEWS DETECTION WITH MACHINE LEARNING**" is a bonafide work carried out by **Melroy Barreto, Jayesh Patel, Utpal Sahakari** towards the partial fulfillment for the award of Bachelor of Engineering in Electronics and Telecommunication Engineering at Padre Conceicao College of Engineering, Verna-Goa during the year 2019-2020. The results contained in this report have not been submitted to any other university or institute for the award of any degree or diploma.

**Dr. Supriya Patil**  
Guide

**Dr. Jayalaxmi Devate**  
Head of the Department

**Dr. Mahesh Parappagoudar**  
Principal

**External Viva**

Name of the Examiners:

Signature with date

# Acknowledgement

We would like to take this opportunity to thank all those involved in the process of making this project a success.

First of all, we would like to thank the principal, Dr. Mahesh Parappagoudar and the head of the Electronics and Telecommunication department, Dr. Jayalaxmi Devate for their cooperation and support.

We would like to express our deepest gratitude to Professor Dr. Supriya Patil, our project supervisor, for her patient guidance, enthusiastic encouragement and useful critiques of this project work. Without her active involvement, this project would not have been possible. We wish to thank our parents for their support and encouragement throughout our study.

# Abstract

News reading on social media becomes more and more popular and fake news becomes a major issue concerning the public and government especially politically. It is essential to flag the fake news before it goes viral and provides misleading information. This Project comes up with the applications of NLP (Natural Language Processing) techniques for detecting the 'fake news', that is, misleading news stories that come from non-reputable sources. Only by building a model based on a count vectorizer (using word tallies) or a (Term Frequency Inverse Document Frequency) TFIDF matrix, (word tallies relative to how often they're used in other articles in your dataset) can only get you so far.

But these models do not consider the important qualities like word ordering and context. It is very possible that two articles that are similar in their word count will be completely different in their meaning. The data science community has responded by taking actions against the problem. There is a Kaggle competition called the "Fake News Challenge " and Facebook is employing AI to filter fake news stories out of users' feeds. Combatting the fake news is a classic text classification project with a straightforward proposition.

Is it possible for you to build a model that can differentiate between "Real" news and "Fake" news? So this is a proposed work on assembling a dataset of both fake and real news and employing different classifiers in order to create a model to classify an article into fake or real based on its words and phrases.

# Contents

<b>1. Introduction</b>	<b>9</b>
1.1. Why This Project? . . . . .	9
1.2. Objective . . . . .	10
1.3. Motivation . . . . .	10
<b>2. Block Diagram</b>	<b>12</b>
<b>3. Design Flow (Overview)</b>	<b>13</b>
3.1. Dataset. . . . .	13
3.2. Pre-processing. . . . .	13
3.3. Feature Extraction. . . . .	13
3.4. Algorithm Training. . . . .	15
3.5. Testing. . . . .	16
<b>4. Analysis</b>	<b>17</b>
4.1. Preprocessing . . . . .	17
4.2. Training Algorithm . . . . .	18
<b>5. Observation and testing</b>	<b>39</b>
<b>6. Results &amp; Conclusion</b>	<b>53</b>
<b>7. Future Scope</b>	<b>54</b>
<b>8. Bibliography</b>	<b>55</b>

# List of Figures

2.1. Block Diagram. . . . .	12
3.1. Confusion matrix. . . . .	16
4.1. Pre-processing output. . . . .	17
4.2. 2-D planar linear model of SVM. . . . .	18
4.3. 3-D planar non-linear model of SVM. . . . .	19
4.4. k-Means clustering flowchart . . . . .	22
4.5. Simple figure of k-means clusters . . . . .	23
4.6. kNN algorithm flowchart . . . . .	24
4.7. Example of kNN implementation with green circle the element to be classified . . . . .	25
4.8. Block level diagram of EBPA . . . . .	26
4.9. A simple visual representation of decision trees . . . . .	27
4.10. The various levels of a decision tree . . . . .	29
4.11. Overfitting of model at a depth of 5 classes . . . . .	29
4.12. Bootstrapping . . . . .	30
4.13. Feature Randomness . . . . .	31
4.14. Gradient Descent vs Conjugate Gradient . . . . .	32
4.15. A-Conjugate Vectors in 2-D Space . . . . .	33
4.16. Method of Orthogonal Direction . . . . .	37
5.1. SVM implementation code . . . . .	39
5.2. SVM confusion matrix . . . . .	40
5.3. kNN implementation code . . . . .	41
5.4. kNN confusion matrix . . . . .	42
5.5. k-Means implementation code . . . . .	43
5.6. k-Means confusion matrix . . . . .	44
5.7. EBPA implementation code . . . . .	45

5.8. EBPA confusion matrix . . . . .	46
5.9. Decision tree implementation code . . . . .	47
5.10. Decision tree confusion matrix . . . . .	48
5.11. Random Forests implementation code . . . . .	49
5.12. Random Forests confusion matrix . . . . .	50
5.13. Conjugate gradient implementation code . . . . .	52
5.14. Conjugate gradient confusion matrix . . . . .	53

# List of Tables

5.1. EBPA with comparison of the training sets based on variation in iteration & learning rate parameters. . . . .	46
6.1. Various Algorithms and their corresponding accuracy. . . . .	53



# Chapter 1

## Introduction

---

### 1.1. Why This Project?

These days' fake news is creating different issues from sarcastic articles to fabricated news and plan government propaganda in some outlets. Fake news and lack of trust in the media are growing problems with huge ramifications in our society. Obviously, a purposely misleading story is "fake news " but lately blathering social media's discourse is changing its definition. Some of them now use the term to dismiss the facts counter to their preferred viewpoints.

The importance of disinformation within American political discourse was the subject of weighty attention , particularly following the American president election . The term 'fake news' became common parlance for the issue, particularly to describe factually incorrect and misleading articles published mostly for the purpose of making money through page views. In this report it is sought to produce a model that can accurately predict the likelihood that a given article is fake news.

The fake news can take advantage of multimedia content to mislead readers and get dissemination, which can cause negative effects or even manipulate the public events. One of the unique challenges for fake news detection on social media is correct identification of fake news on newly emerged events. Unfortunately, most of the existing approaches can hardly handle this challenge, since they tend to learn event-specific features that can not be transferred to unseen events.

The main issue in carrying out this work is that real time news cannot be distinguished or perhaps the trusted source itself might give fake news at most periods of time. Hence the sources should be surveyed for a particular period of time and also at the same time broaden the coverage of sources.

Facebook has been at the epicenter of much critique following media attention. They have already implemented a feature to flag fake news

on the site when a user sees it ; they have also said publicly they are working on distinguishing these articles *in* an automated way. Certainly, it is not an easy task. A given algorithm must be politically unbiased – since fake news exists on both ends of the spectrum – and also give equal balance to legitimate news sources on either end of the spectrum. In addition, the question of legitimacy is a difficult one, However in order to solve this problem, it is necessary to have an understanding on what Fake News is. Later, it is needed to look into how the techniques in the fields of machine learning, natural language processing help us to detect fake news.

## **1.2. Objective**

The main objective is to detect the fake news, which is a classic text classification problem with a straightforward proposition. It is needed to build a model that can differentiate between “Real” news and “Fake” news.

## **1.3. Motivation**

Social media for news consumption is a double-edged sword. On the one hand, its low cost, easy access, and rapid dissemination of information lead people to seek out and consume news from social media. On the other hand, it enables the wide spread of "fake news", i.e., low quality news with intentionally false information. The extensive spread of fake news has the potential for extremely negative impacts on individuals and society. Therefore, fake news detection on social media has recently become an emerging research that is attracting tremendous attention. Fake news detection on social media presents unique characteristics and challenges that make existing detection algorithms from traditional news media ineffective or not applicable. First, fake news is intentionally written to mislead readers to believe false information, which makes it difficult and nontrivial to detect based on news content; therefore, we need to include auxiliary information, such as user social engagements on social media, to help

make a determination. Second, exploiting this auxiliary information is challenging in and of itself as users' social engagements with fake news produce data that is big, incomplete, unstructured, and noisy. Because the issue of fake news detection on social media is both challenging and relevant, we conducted this survey to further facilitate research on the problem. In this survey, we present a comprehensive review of detecting fake news on social media, including fake news characterizations on psychology and social theories, existing algorithms from a data mining perspective, evaluation metrics and representative datasets. We also discuss related research areas, open problems, and future research directions for fake news detection on social media.

# Chapter 2

## Block Diagram

---

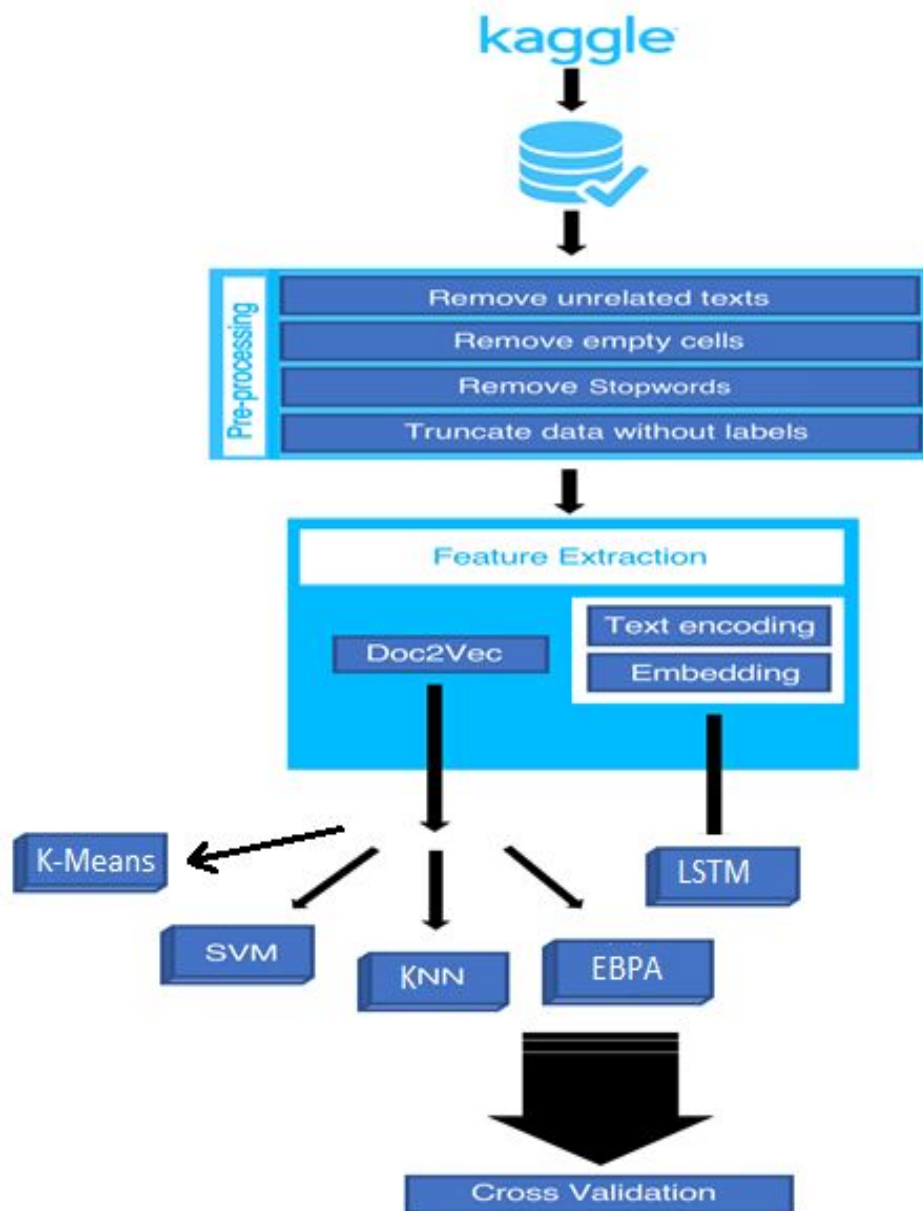


Fig 2.1: Block Diagram

# Chapter 3

## Design Flow (Overview)

---

### 3.1. Dataset

**Kaggle**, a subsidiary of [Google doc](#), is an online community of [data scientists](#) and [machine learning](#) practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Hence most of the datasets are collected from Kaggle and then put through for processing.

### 3.2. Pre-processing

It involves processing of data for further analysis. This generally includes vector representation of data i.e. Words and sentences. In this process various unnecessary data such as non-character symbols as well as adjectives, pronoun are removed for easy and faster processing.

It involves processing through various application plugins as well as library functions. Now the dataset is ready to be evaluated for algorithmic test and data processing.

### 3.3. Feature Extraction

Doc2Vec is a model developed in 2014 based on the existing Word2Vec model, which generates vector representations for words. Word2Vec represents documents by combining the vectors of the individual words, but in doing so it loses all word order information.

Doc2Vec expands on Word2Vec by adding a ‘document vector’ to the output representation, which contains some information about the document as a whole, and allows the model to learn some information about word order. Preservation of word order information makes Doc2Vec useful for our application, as it aims to detect subtle differences between text documents.

The representation of words as vectors can be either done using one-hot encodings where in each word is represented as a zero vector with length equal to the vocabulary, then adding one to the index of corresponding position of the word, consider words ‘cat’, ‘sat’, ‘on’, ‘mat’ hence the vector representation for ‘cat’ will be 1000 and so on, or another approach is to encode each word using a unique number. Consider the same words ‘cat’, ‘sat’, ‘on’, ‘mat’, we could assign 1 to ‘cat’, 2 to ‘mat’, and so on. And then encode the sentence ‘cat sat on mat’ as a dense vector like [1, 2, 4, 3].

These two representations techniques were however not used because of their apparent shortcomings. Instead a better approach was ‘word embeddings’. Word embeddings give us a way to use an efficient, dense representation in which similar words have a similar encoding. An embedding is a dense vector of floating point values (the length of the vector is a parameter you specify). It is common to see word embeddings that are 8-dimensional (for small datasets), up to 1024-dimensions when working with large datasets.

The Word2Vec model is divided into two types:

### 3.3.1 Cbow

Continuous bag of words creates a sliding window around the current word and predicts it from “context” — the surrounding words. Each word is represented as a feature vector. After training, these vectors become the word vectors.

### 3.3.2 Skip Gram

This algorithm is just the opposite of CBOW. Instead of predicting one word, we use 1 word to predict all the surrounding words. It is

slower than CBOW but considered more accurate with infrequent words.

Doc2Vec is just an addition of ‘paragraph id’. This ‘paragraph id’ is used to train the Doc2Vec model. There are two approaches within Doc2Vec: dbow and dmpv.

dbow works in the same way as skip-gram, except that the input is replaced by a special token representing the document. In this architecture, the order of words in the document is ignored; hence the name distributed bag of words.

dmpv works in a similar way to cbow. For the input, dmpv introduces an additional document token in addition to multiple target words. Unlike cbow, however, these vectors are not summed but concatenated. The objective is again to predict a context word given the concatenated document and word vectors.

### **3.4. Algorithmic Training**

Here after processing, data is processed for various algorithms namely

- Support Vector Machine (SVM)
- k-Nearest Neighbors (kNN)
- k-Means Clustering
- Error Back Propagation Algorithm (EBPA)
- Decision Tree
- Random Forest
- Conjugate Gradient

### 3.5. Testing

Once a data set is trained it is tested through various algorithms and an output is presented in the form of a **confusion matrix**.

**Confusion matrix** is a performance measurement for machine learning classification problems where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

Ex-

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Fig 3.1: Confusion Matrix



# Chapter 4

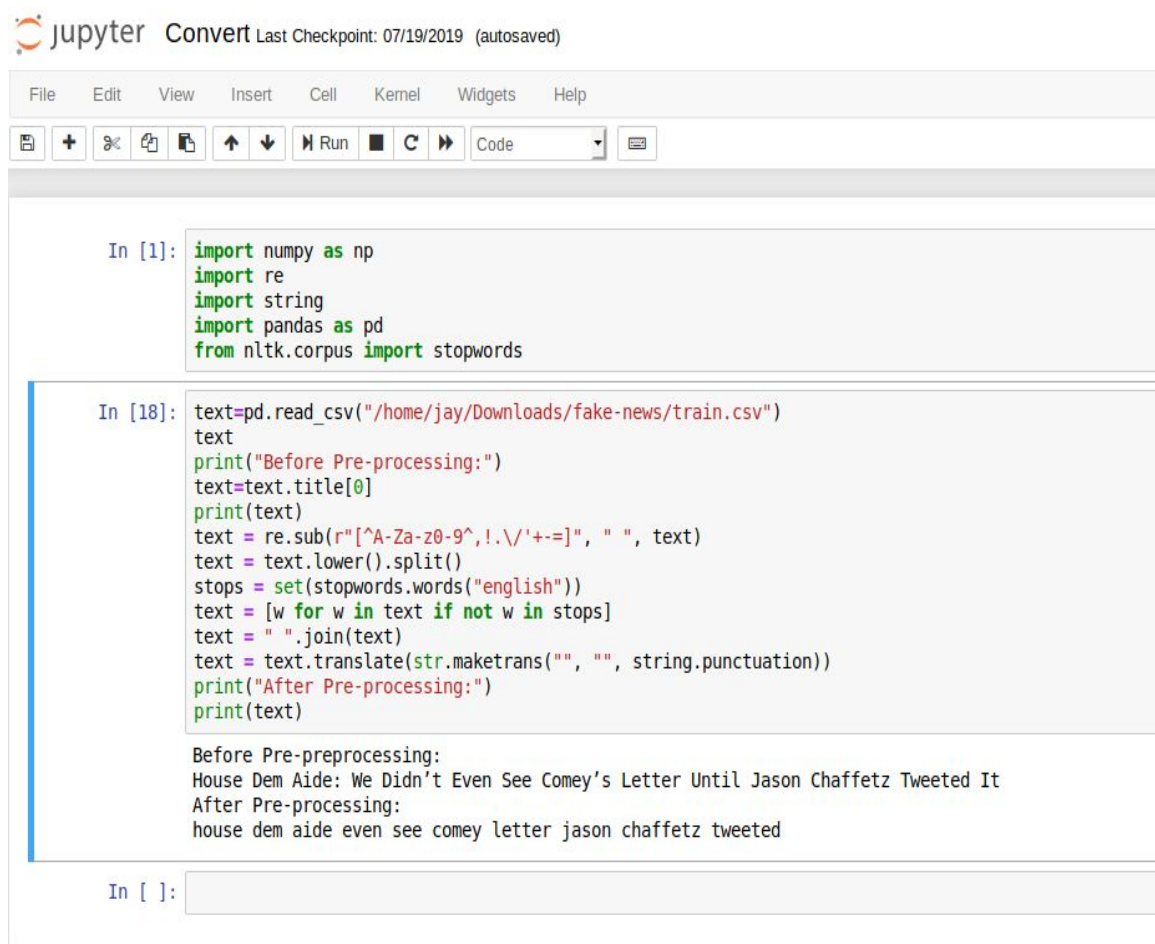
## Analysis

---

### 4.1. Pre-processing

Example of how pre-processing works-

-before processing and after processing of sentence is covered in the following code



The image shows a Jupyter Notebook interface. At the top, it says 'jupyter Convert Last Checkpoint: 07/19/2019 (autosaved)'. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Under the menu bar is a toolbar with icons for saving, adding cells, running, and other functions. The main area contains two code cells. The first cell, labeled 'In [1]:', contains import statements for numpy, re, string, pandas, and nltk.corpus stopwords. The second cell, labeled 'In [18]:', contains code to read a CSV file, print the title, convert to lowercase, remove stopwords, and remove punctuation. Below the code in the second cell, the output is displayed: 'Before Pre-preprocessing: House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It' and 'After Pre-processing: house dem aide even see comey letter jason chaffetz tweeted'.

```
In [1]: import numpy as np
import re
import string
import pandas as pd
from nltk.corpus import stopwords

In [18]: text=pd.read_csv("/home/jay/Downloads/fake-news/train.csv")
text
print("Before Pre-processing:")
text=text.title[0]
print(text)
text = re.sub(r"[^A-Za-z0-9^,!\./'++=]", " ", text)
text = text.lower().split()
stops = set(stopwords.words("english"))
text = [w for w in text if not w in stops]
text = " ".join(text)
text = text.translate(str.maketrans("", "", string.punctuation))
print("After Pre-processing:")
print(text)

Before Pre-preprocessing:
House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It
After Pre-processing:
house dem aide even see comey letter jason chaffetz tweeted

In [ ]:
```

Fig 4.1: Pre-processing output

## 4.2 Training Algorithms

### 4.2.1 Support Vector Machine (SVM)

The support vector machine algorithm is oriented with the objective to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the characteristic data points.

Firstly we need to understand certain keywords used across the algorithm.

- **Hyperplane** – It is a decision plane or space which separates the character set into different classes.
- **Support Vectors** – Data Points that are closest to the hyperplane are called support vectors. Separating lines/Hyperplanes will be defined with the help of these data points.
- **Margin** – It may be defined as the gap between two closest data points of different classes. It can be calculated as the perpendicular distance from the hyperplane to the support vectors. Large margin is considered as a good data classification and small margin is considered as a dense data classification.

*2-D Plane (Linear)*

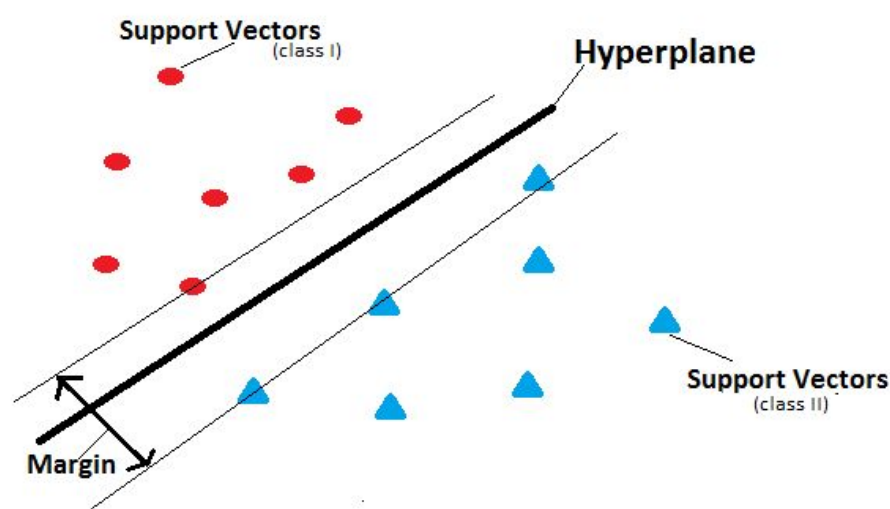


Fig 4.2: 2-D planar linear model of SVM

To separate the data points into two classes, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between characteristic data points of both classes. Maximizing the margin distance between them can enable us to classify the input data with more confidence.

Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line or linear. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane and dimensionality increases with an increased number of input features.

Kernel is used for data points separation and to determine the hyperplane.

### *3-D plane (Non-linear)*

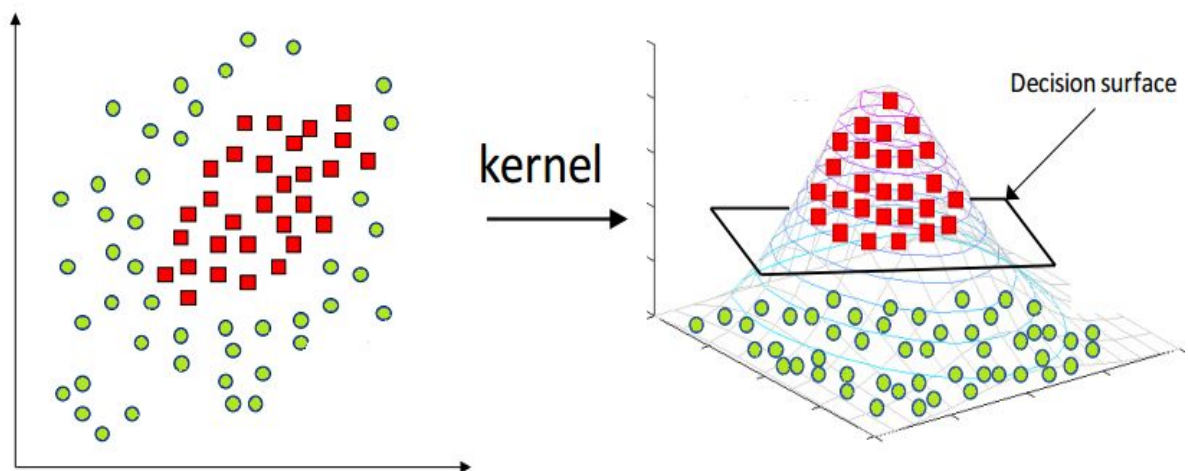


Fig 4.3: 3-D planar non-linear model of SVM

#### *4.2.1.1 Linear Kernel*

It can be used as a dot product between any two observations. The formula of linear kernel is as below –

$$K(x, xi) = \sum (x * xi)$$

We can see that the product between two vectors say  $x$  &  $xi$  is the sum of the multiplication of each pair of input values.

A more generalized form of linear kernel and distinguishes curved or nonlinear input space. Following is the formula for polynomial kernel-

$$K(x, xi) = 1 + \sum (x * xi)^d \quad K(x, xi) = 1 + \sum (x * xi)^d$$

Here  $d$  is the degree of polynomial, which we need to specify manually in the learning algorithm.

#### 4.2.1.2 Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically –

$$K(x, xi) = \exp(-\gamma * \sum (x - xi)^2) \quad K(x, xi) = \exp(-\gamma * \sum (x - xi)^2)$$

Here,  $\gamma$  ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of  $\gamma$  is 0.1.

## 4.2.2 k-Means Clustering

Clustering is one of the most common exploratory data analysis techniques used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. K means clustering is one of the most widely used clustering algorithms, which is widely used.

K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume  $k$  clusters) fixed apriori. The main idea is to define  $k$  centers, one for each cluster. These centers should be placed in a cunning way because different locations cause different results. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate  $k$  new centroids as barycenters of the clusters resulting from the previous step. After we have these  $k$  new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the  $k$  centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^c \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where, ' $\|x_i - v_j\|$ ' is the Euclidean distance between  $x_i$  and  $v_j$ .

' $c_i$ ' is the number of data points in the cluster.

' $c$ ' is the number of cluster centers.

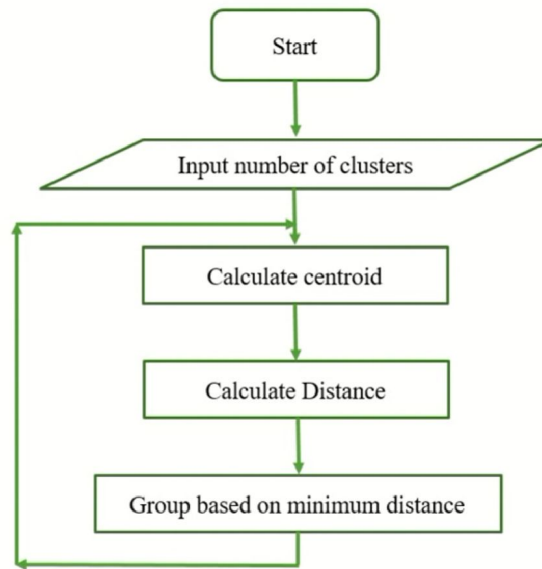


Fig 4.4: k-Means clustering flowchart

### Algorithmic steps for k-means clustering

Let  $X = \{x_1, x_2, x_3, \dots, x_n\}$  be the set of data points and

$V = \{v_1, v_2, \dots, v_c\}$  be the set of centers.

- 1) Randomly select 'c' cluster centers.
- 2) Calculate the distance between each data point and cluster centers.
- 3) Assign the data point to the cluster center whose distance from the cluster center is the minimum of all the cluster centers..
- 4) Recalculate the new cluster .
- 5) Recalculate the distance between each data point and new obtained cluster centers.
- 6) If no data point was reassigned then stop, otherwise repeat from step 3).

## Example of how K means clustering works -

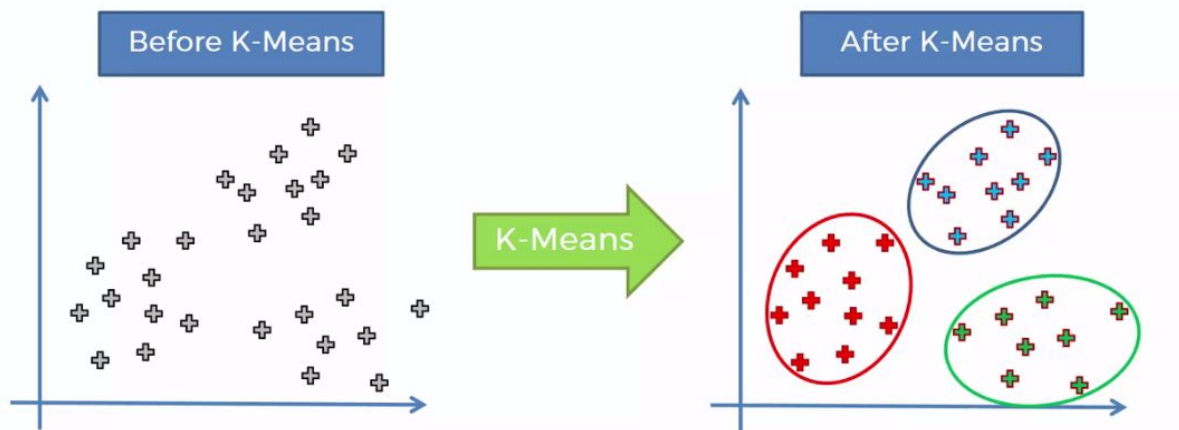


Fig 4.5: Simple figure of k-means clusters

### 4.2.3 k-Nearest Neighbors

The k-nearest neighbors (kNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. It is a non-parametric, lazy algorithm. Its purpose is to use a database of many classified points to predict the classification of new sample points. It is also a lazy algorithm. What this means is that it does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal. This also means that the training phase is pretty fast. Lack of generalization means that kNN keeps all the training data. To be more exact, all (or most) the training data is needed during the testing phase. The k value should always be chosen as an odd value.

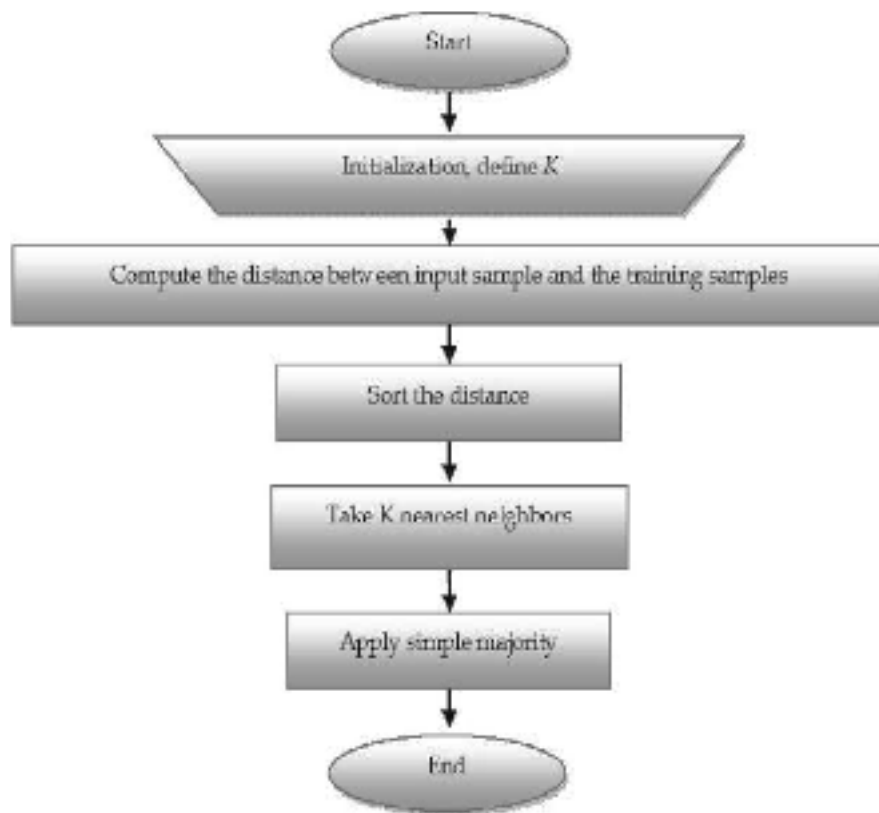


Fig 4.6: kNN algorithm flowchart

The kNN Algorithm:

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
4. Calculate the distance between the query example and the current example from the data.
5. Add the distance and the index of the example to an ordered collection
6. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
7. Pick the first K entries from the sorted collection



8. Get the labels of the selected K entries
9. If regression, return the mean of the K label.
10. If classification, return the mode of the K labels

### **Example of kNN -**

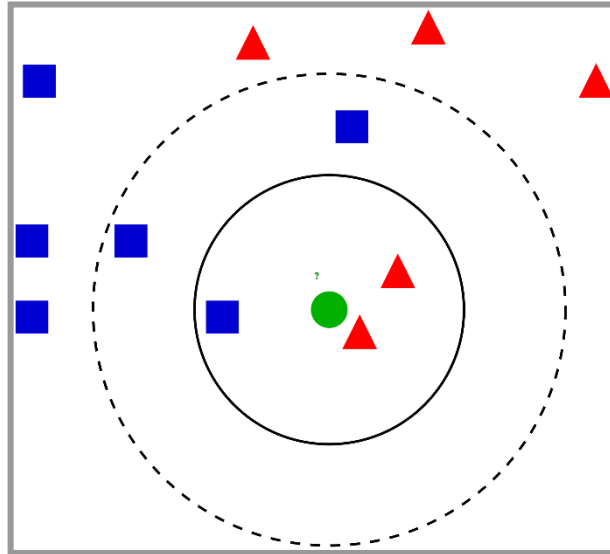


Fig 4.7: Example of kNN implementation with green circle the element to be classified

### **4.2.4 EBPA**

It is an error reducing algorithm used in artificial neural networks. Artificial neural networks are networks based on the human's nerve system. These networks contain well defined sets of inputs and outputs. The network is used to describe the complex relationship between the inputs and outputs of the network. The name comes from the complexity of the network because the human's nervous system is so much complex which is known by all of us.

Algorithm working:

1. Calculated the feed-forward signals from the input to the output.
2. Calculate output error  $E$  based on the predictions  $\hat{t}_k$  and the target  $t_k$
3. Back Propagate the error signals by weighing it by the weights in previous layers and the gradients of the associated activation functions
4. Calculating the gradients  $\frac{\partial E}{\partial \theta}$  for the parameters based on the back propagated error signal and the feedforward signals from the inputs.
5. Update the parameters using the calculated gradients

$$\theta \leftarrow \theta - \eta \frac{\partial E}{\partial \theta}$$

## Block Level Implementation

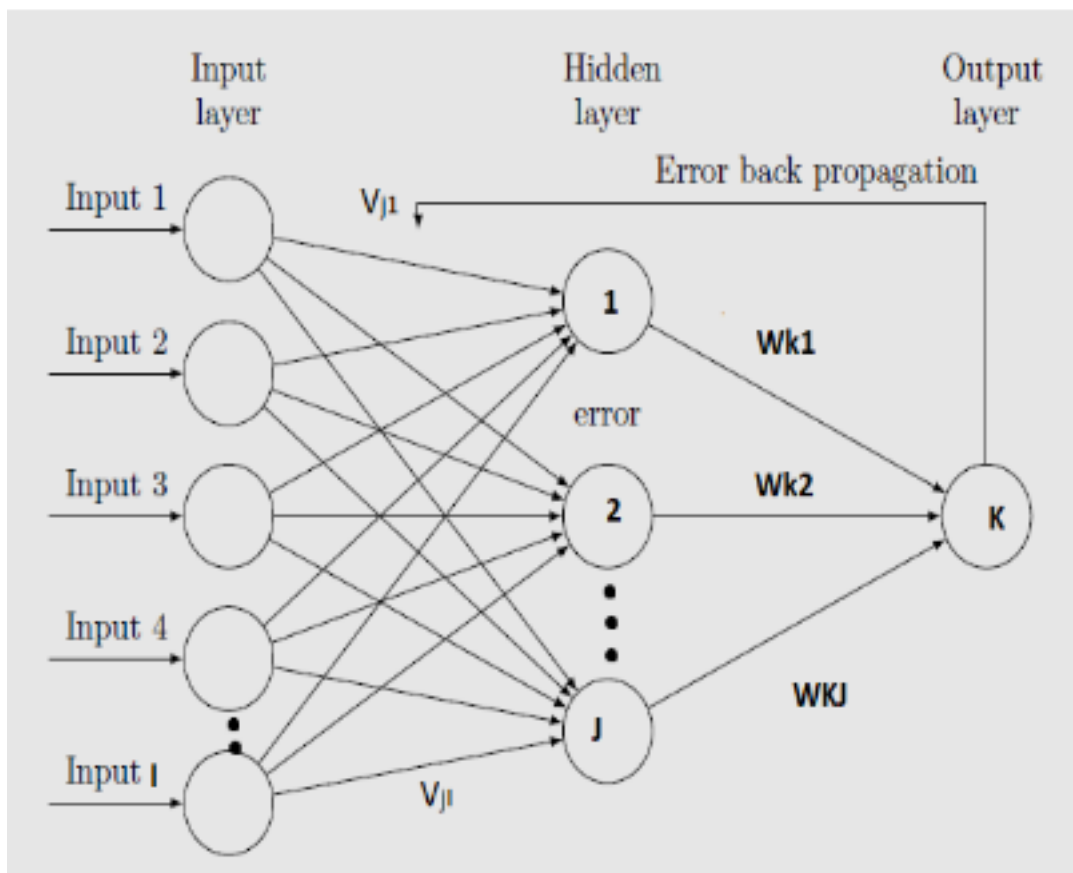


Fig 4.8: Block level diagram of EBPA

## 4.2.5 Decision Trees

A decision tree builds upon iteratively asking questions to partition data. It is easier to conceptualize the partitioning data with a visual representation of a decision tree:

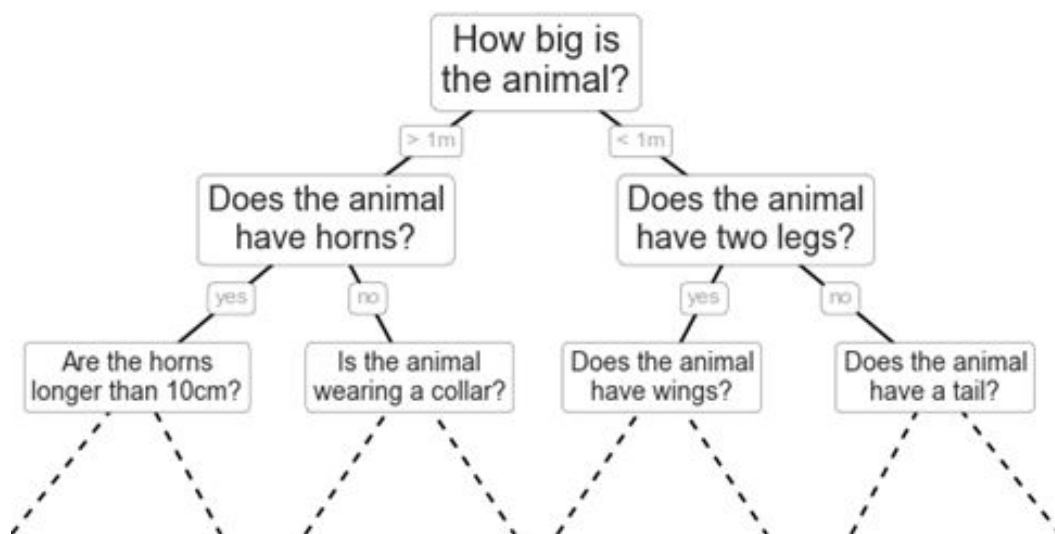


Fig 4.9: A simple visual representation of Decision Tree

This represents a decision tree to classify animals. First split is based on the size of the animal. Although the question seems to be “How big is the animal?”, it is asked in the form of “Is the animal bigger than 1m?” because we want to split the data points in two groups at each step. The questions get more specific as the tree gets deeper.

What you ask at each step is the most critical part and greatly influences the performance of decision trees. For example, assume your dataset has “feature A” ranging from 0 to 100 but most of the values are above 90. In this case, the first question to ask is “Is feature A more than 90?”. It does not make sense to ask “Is feature A more than 50?” because it will not give us much information about the dataset.

There are two ways to measure the quality of a split: **Gini Impurity** and **Entropy**. They essentially measure the impurity and give similar results. Scikit-learn uses the gini index by default but you can change it to entropy using criterion parameters.

## *Gini Impurity*

As stated on wikipedia, “Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset”. It basically means that impurity increases with randomness. For instance, let’s say we have a box with ten balls in it. If all the balls are the same color, we have no randomness and impurity is zero. However, if we have 5 blue balls and 5 red balls, impurity is 1.

## *Entropy and Information Gain*

Entropy is a measure of uncertainty or randomness. The more randomness a variable has, the higher the entropy is. The variables with uniform distribution have the highest entropy. For example, rolling a fair dice has 6 possible outcomes with equal probabilities so it has a uniform distribution and high entropy.

Splits that result in more pure nodes are chosen. All these indicate “information gain” which is basically the difference between entropy before and after the split.

When choosing a feature to split, decision tree algorithm tries to achieve

- More predictiveness
- Less impurity
- Lower entropy

The next topic is the number of questions. How many questions do we ask? When do we stop? When is our tree sufficient to solve our classification problem? The answer for all these questions lead us to one of the most important concepts of machine learning: overfitting. The model can keep asking questions until all the nodes are pure. Pure nodes include data points from only one class.

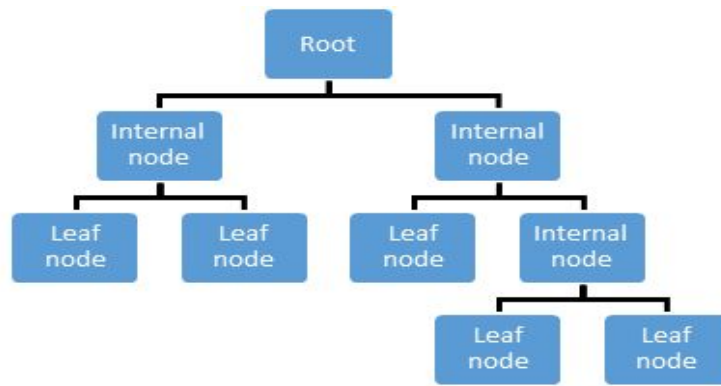


Fig 4.10: The various levels of a decision tree

As you can see in the visualization below, at a depth of 5, the model clearly overfits. The narrow regions between classes might be due to outliers or noise.

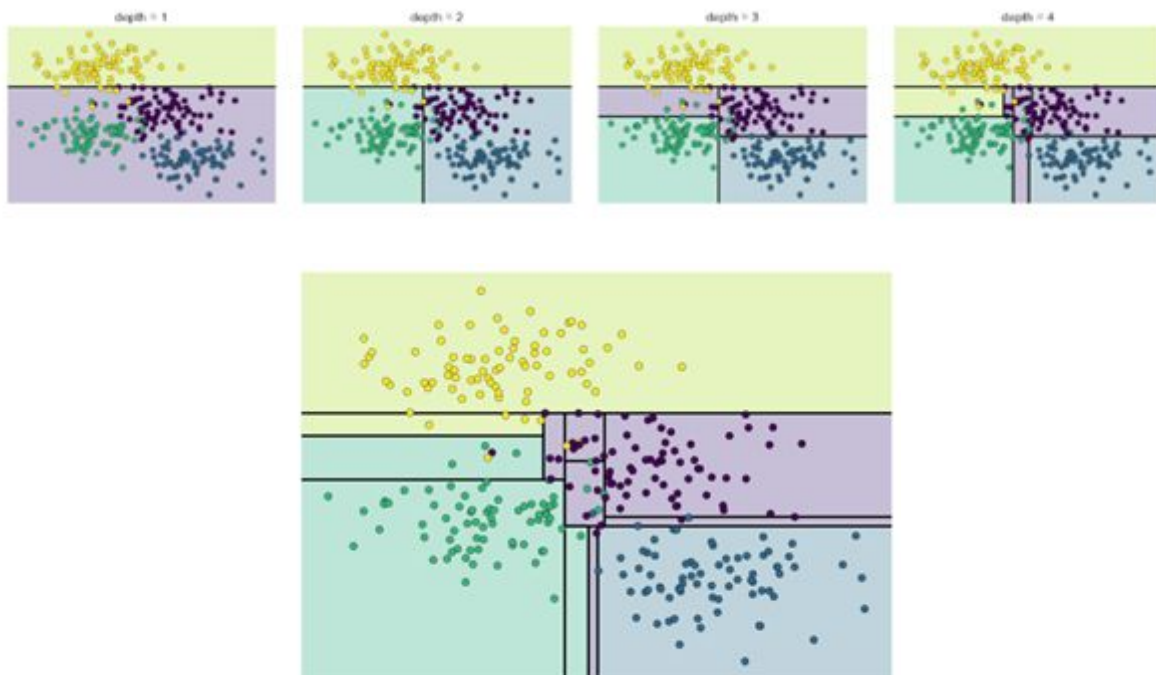


Fig 4.11: Overfitting of model at a depth of 5 classes

## 4.2.6 Random Forests

Random forest is an ensemble of many decision trees. Random forests are built using a method called bagging in which each decision tree is used as parallel estimators. If used for a classification problem, the

result is based on the majority vote of the results received from each decision tree. For regression, the prediction of a leaf node is the mean value of the target values in that leaf. Random forest regression takes the mean value of the results from decision trees.

Random forests reduce the risk of overfitting and accuracy is much higher than a single decision tree. Furthermore, decision trees in a random forest run in parallel so that the time does not become a bottleneck.

The success of a random forest highly depends on using uncorrelated decision trees. If we use the same or very similar trees, the overall result will not be much different than the result of a single decision tree. Random forests achieve uncorrelated decision trees by **bootstrapping** and **feature randomness**.

- Bootstrapping is randomly selecting samples from training data with replacement. They are called bootstrap samples. The following figure clearly explains this process:



Fig 4.12: Bootstrapping

- Feature randomness is achieved by selecting features randomly for each decision tree in a random forest. The number of features used for each tree in a random forest can be controlled with the `max_features` parameter.

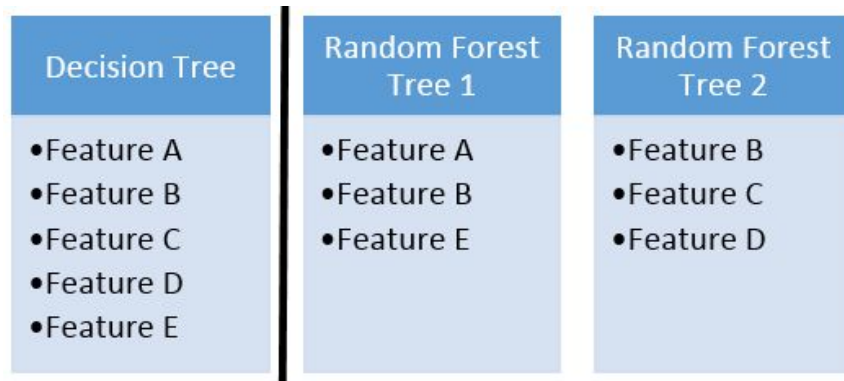


Fig 4.13: Feature Randomness

Bootstrap samples and feature randomness provide the random forest model with uncorrelated trees.

There is an additional parameter introduced with random forests:

**n\_estimators**: Represents the number of trees in a forest. To a certain degree, as the number of trees in a forest increases, the result gets better. However, after some point, adding additional trees does not improve the model. Please keep in mind that adding additional trees always means more time for computation.

### 4.2.7 Conjugate Gradient

Conjugate Gradient is an extension of steepest gradient descent. For the steepest gradient, we step in one direction per iteration. Through the iterations, we found that the new directions may contain the component of the old directions and the process walks in zig-zag patterns. For conjugate gradients, we consider multiple directions simultaneously. Hence, we avoid repeating the old directions. In 1952, Hestenes and Stiefel independently introduced conjugate gradient formulas to simplify the multiple direction search.

The Conjugate Gradient (CG) method is used to solve a linear equation or to optimize a quadratic equation. It is more efficient in solving those problems than the gradient descent.

Solving  $Ax = b$

is equivalent to

$$\underset{x}{\text{maximize}} \quad f(x) = \frac{1}{2}x^T Ax - b^T x$$

$$\text{since } f'(x) = Ax - b = 0$$

$$\mathbf{x}^T A \mathbf{x} > 0$$

where  $A$  is symmetrical and positive definite.

In a line search, we determine the steepest direction of ascent and then select the step size. For example, in the gradient ascent method, we take a step size equal to the gradient times the learning rate. For the figure on the left below, the deepest direction according to the gradient contour (the dotted ellipse) is moving right. The next move may go up and slightly to the left according to the steepest gradient at the current point.

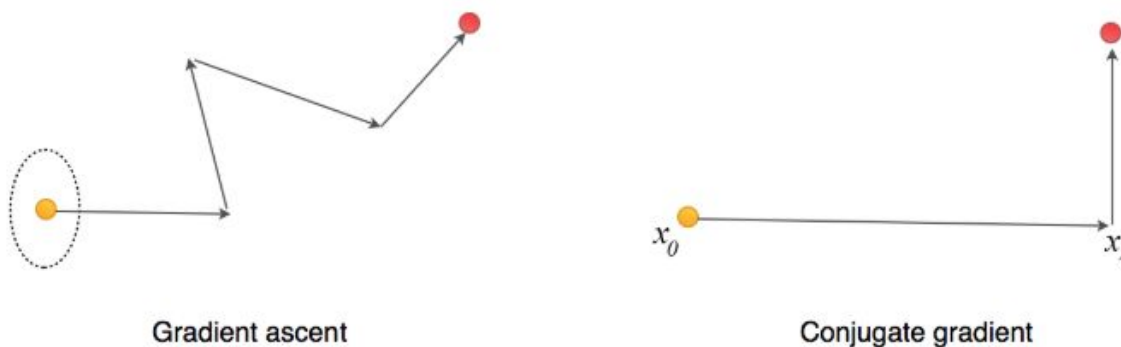


Fig 4.14: Gradient Descent vs Conjugate Gradient



The conjugate gradient method is a line search method but for every move, it would not undo part of the moves done previously. It optimizes a quadratic equation in fewer step than the gradient ascent. If  $x$  is  $N$ -dimensional ( $N$  parameters), we can find the optimal point in at most  $N$  steps. For every move, we want a direction conjugate to all previous moves. This guarantees that we do not undo part of the moves we did. In short, if  $x$  is 4-dimensional, it should take at most 4 moves to reach the optimal point.

1. We start the ascent in one particular direction.
2. We settle down in the optimal point for that direction.
3. We find a new direction  $d_j$  which is A-conjugate to any previous moving directions  $d_i$ .

Mathematically, it means any new direction  $d_j$  must obey the A-conjugate with all previous direction  $d_i$ :

$$d_{(i)}^T A d_{(j)} = 0.$$

where  $A$  is the matrix in the quadratic equation. Here are some other examples of A-conjugate vectors in 2-D space.

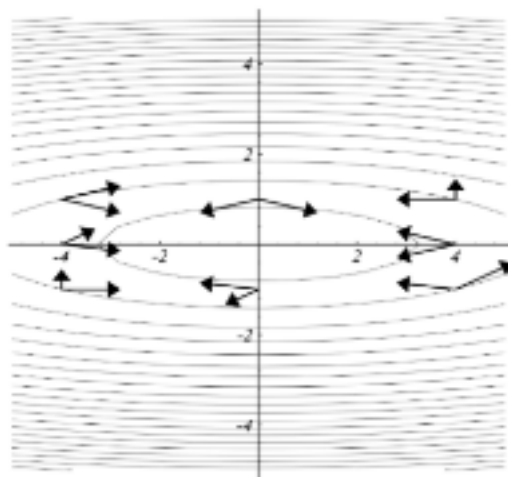


Fig 4.15: A-Conjugate Vectors in 2-D Space

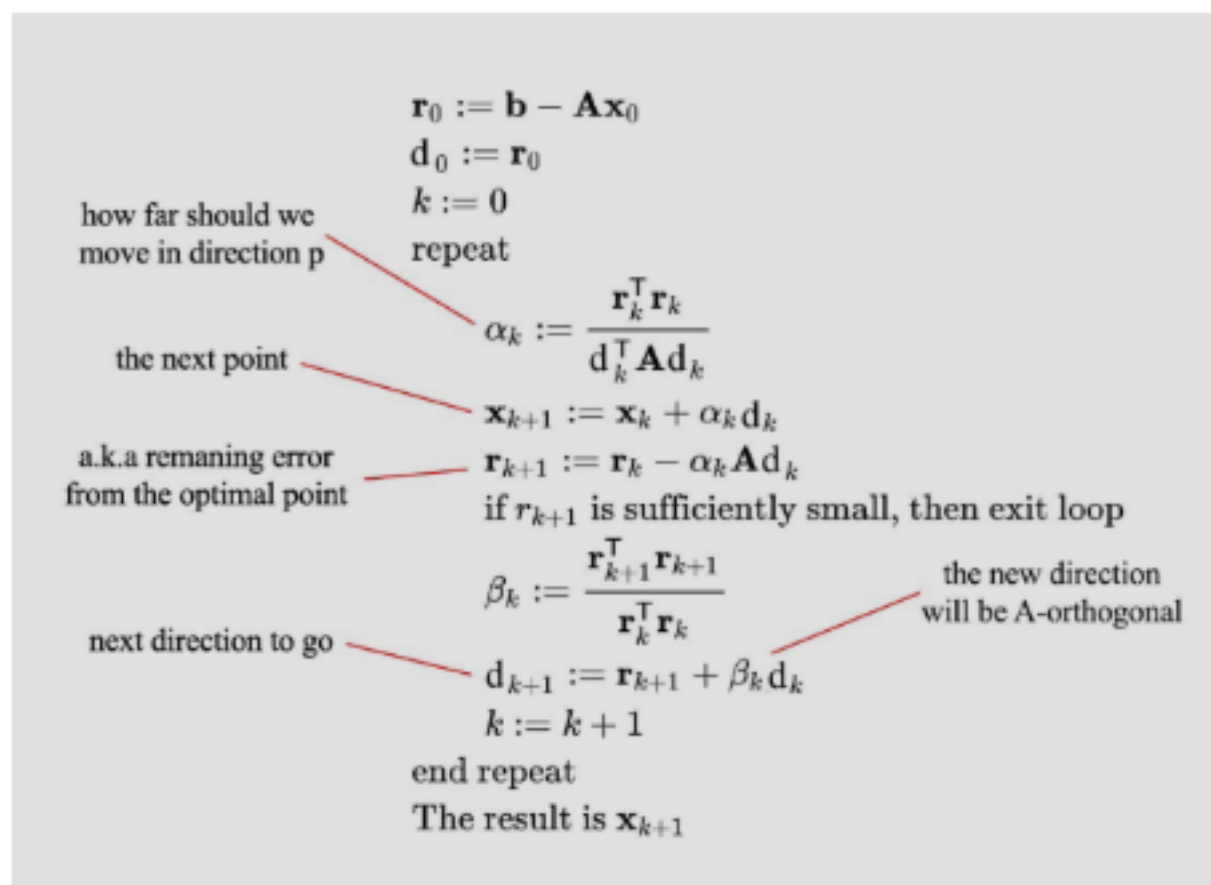
The A-conjugate vectors are independent of each other. Hence, N A-conjugate vectors can span a space of N dimensions.

$$\mathbf{Ax}_* = \sum_{i=1}^n \alpha_i \mathbf{Ad}_i$$

The key part of the CG method is to find  $\alpha$  and  $d$ .

## CG Algorithm

But let's have a preview of the algorithm first. We start with a random (or educated) guess for the solution on  $X$  ( $x_0$ ) and calculate the next guess  $x_1$  with  $\alpha$  and  $d$ .



$d$  is the direction (the conjugate vector) to go for the next move.

Let's see how it works in detail. First, we define two terms:

- $e$  is the error between the current guess and the optimal point.



Conjugate gradient

- $r$  measures how far we are from the correct value  $b$  ( $Ax=b$ ). We can view  $r$  as the error  $e$  transformed by  $A$  to the space of  $b$  (how far  $Ax$  is from  $b$ ).

$$e_{(i)} = x_{(i)} - x$$

$$r_{(i)} = b - Ax_{(i)}$$

From,

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

$$f'(x) = Ax - b$$

We can drive:

$$\begin{aligned}
 -f'(x_{(i)}) &= b - Ax_{(i)} \\
 r_{(i)} &= -Ae_{(i)} && \text{using } \begin{aligned} e_{(i)} &= x_{(i)} - x \\ r_{(i)} &= b - Ax_{(i)} \end{aligned} \\
 r_{(i)} &= -f'(x_{(i)})
 \end{aligned}$$

Next point is computed as (which  $\alpha$  is a scalar and  $d$  is the direction):

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)}.$$

In order for the future direction not to undo progress from the previous directions, let's try  $d$  and  $e$  to be orthogonal. i.e. the remaining error after taking this step should be orthogonal to the current direction. This makes sense if the remaining moves should not undo part of the move we just did.

$$\begin{aligned}
 d_{(i)}^T e_{(i+1)} &= 0 \\
 d_{(i)}^T (e_{(i)} + \alpha_{(i)} d_{(i)}) &= 0 \\
 \alpha_{(i)} &= -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}}.
 \end{aligned}$$

$\alpha$  depends on  $e$  which we don't know its value. So instead of orthogonal, let's try another guess. A new searching direction should be  $A$ -orthogonal with the previous one. The definition of  $A$ -orthogonal is:

$$\begin{aligned}
 d_{(i)}^T e_{(i+1)} &= 0 \\
 d_{(i)}^T (e_{(i)} + \alpha_{(i)} d_{(i)}) &= 0 \\
 \alpha_{(i)} &= -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}}.
 \end{aligned}$$

To fulfill these conditions, the next point  $x_i$  has to be the optimal point in the search direction  $d$ .

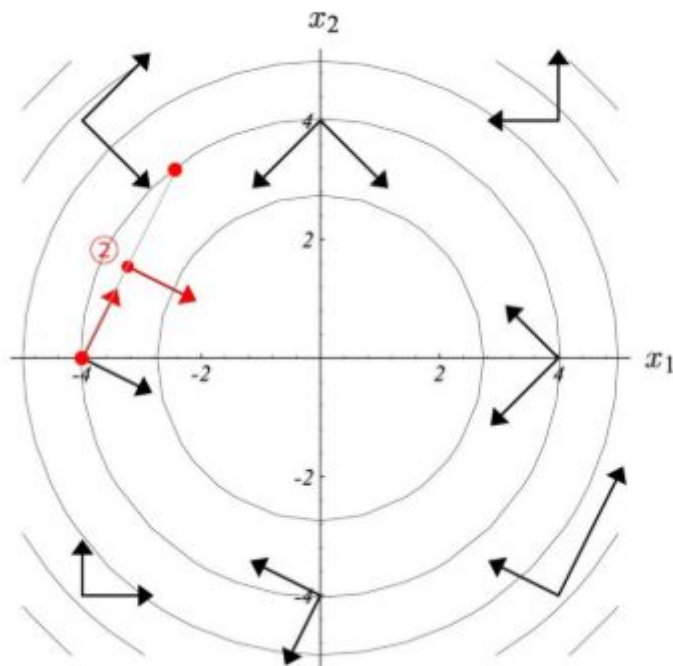


Fig 4.16: Method of Orthogonal Direction

$$\begin{aligned}
\frac{d}{d\alpha} f(x_{(i+1)}) &= 0 \\
f'(x_{(i+1)})^T \frac{d}{d\alpha} x_{(i+1)} &= 0 \\
-r_{(i+1)}^T d_{(i)} &= 0 \\
d_{(i)}^T A e_{(i+1)} &= 0.
\end{aligned}$$

Using the  $A$ -orthogonal requirement,  $\alpha$  is computed as:

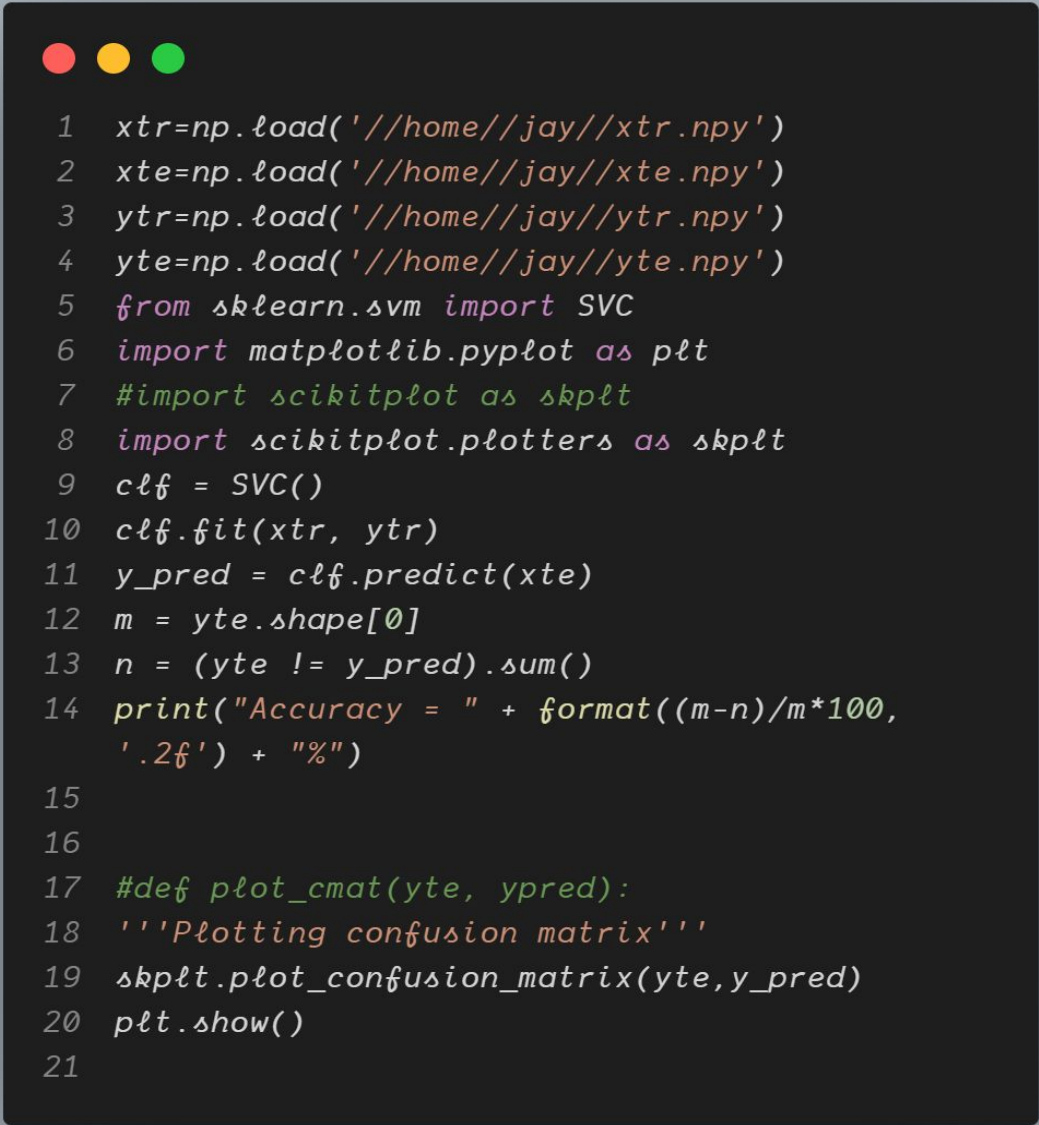
$$\begin{aligned}
\alpha_{(i)} &= -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} \quad \text{apply } r_{(i)} = -A e_{(i)} \\
&= \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}.
\end{aligned}$$

# Chapter 5

## Observations of testing various algorithms

---

### 5.1 Support Vector Machine (SVM)



```
1  xtr=np.load('//home//jay//xtr.npy')
2  xte=np.load('//home//jay//xte.npy')
3  ytr=np.load('//home//jay//ytr.npy')
4  yte=np.load('//home//jay//yte.npy')
5  from sklearn.svm import SVC
6  import matplotlib.pyplot as plt
7  #import scikitplot as skplt
8  import scikitplot.plotters as skplt
9  clf = SVC()
10 clf.fit(xtr, ytr)
11 y_pred = clf.predict(xte)
12 m = yte.shape[0]
13 n = (yte != y_pred).sum()
14 print("Accuracy = " + format((m-n)/m*100,
15     '.2f') + "%")
16
17 #def plot_cmat(yte, ypred):
18     '''Plotting confusion matrix'''
19     skplt.plot_confusion_matrix(yte,y_pred)
20     plt.show()
21
```

Fig 5.1: SVM implementation code

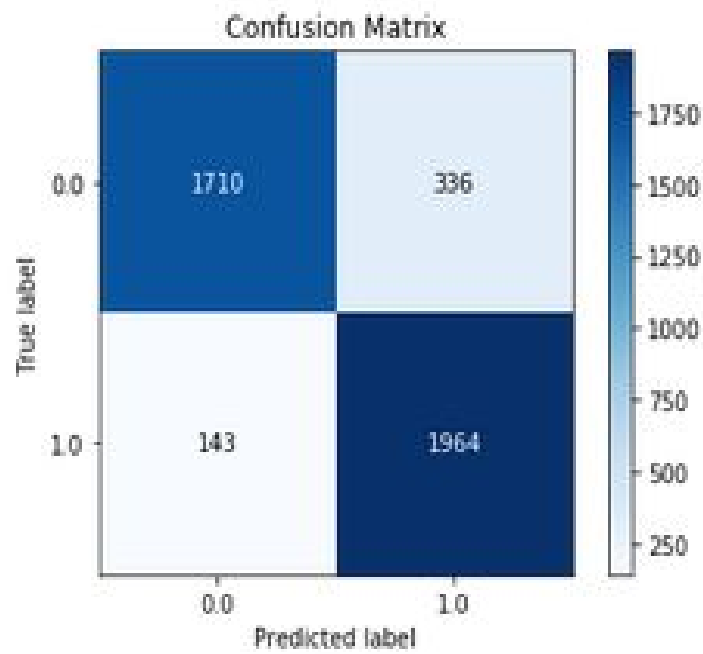


Fig 5.2: SVM confusion matrix

**Accuracy obtained after implementation -**

**Accuracy: 88.47%**



## 5.2 K-Nearest Neighbors

```
1 xtr=np.load('//home//jay//xtr.npy')
2 xte=np.load('//home//jay//xte.npy')
3 ytr=np.load('//home//jay//ytr.npy')
4 yte=np.load('//home//jay//yte.npy')
5 from sklearn.neighbors import KNeighborsClassifier as kn
6 import matplotlib.pyplot as plt
7 #import scikitplot as skplt
8 import scikitplot.plotters as skplt
9 clf = kn()
10 clf.fit(xtr, ytr)
11 y_pred = clf.predict(xte)
12 m = yte.shape[0]
13 n = (yte != y_pred).sum()
14 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
15
16
17 #def plot_cmat(yte, ypred):
18 '''Plotting confusion matrix'''
19 skplt.plot_confusion_matrix(yte,y_pred)
20 plt.show()
21
```

Fig 5.3: kNN implementation code

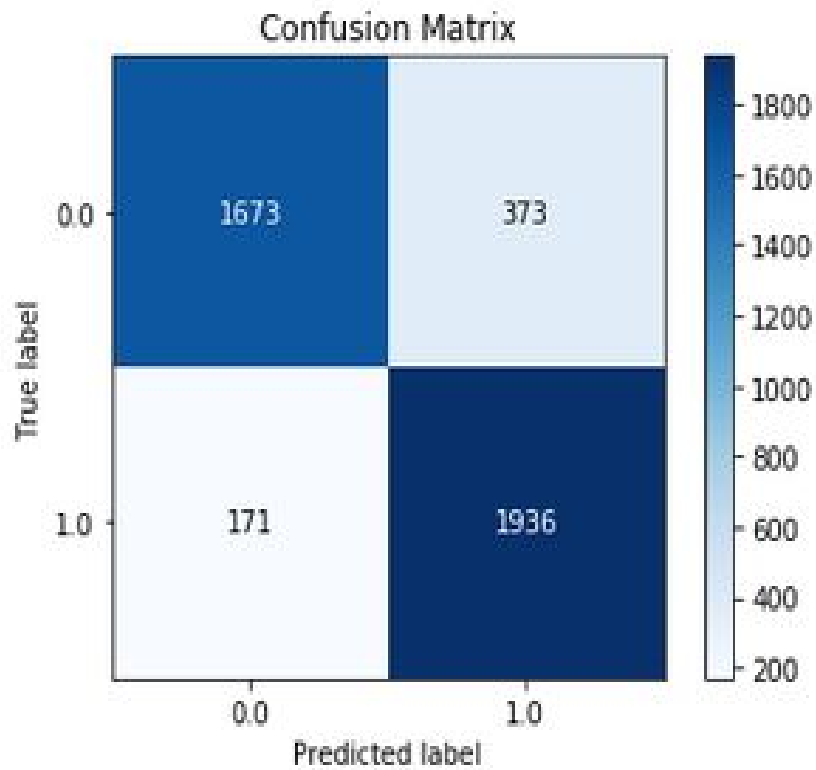


Fig 5.4: kNN confusion matrix

**Accuracy obtained after implementation -**  
**Accuracy: 86.90%**

## 5.3 K-Means Clustering

```
1 xtr=np.load('//home//jay//xtr.npy')
2 xte=np.load('//home//jay//xte.npy')
3 ytr=np.load('//home//jay//ytr.npy')
4 yte=np.load('//home//jay//yte.npy')
5
6 from sklearn.cluster import KMeans as KM
7 import matplotlib.pyplot as plt
8 #import scikitplot as skplt
9 import scikitplot.plotters as skplt
10 clf = KM(n_clusters=2,random_state=0).fit(xtr)
11 #clf.fit(xtr, ytr)
12 y_pred = clf.predict(xte)
13 m = yte.shape[0]
14 n = (yte != y_pred).sum()
15 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
16
17
18 #def plot_cmat(yte, ypred):
19 '''Plotting confusion matrix'''
20 skplt.plot_confusion_matrix(yte,y_pred)
21 plt.show()
```

Fig 5.5: k-Means Clustering implementation code

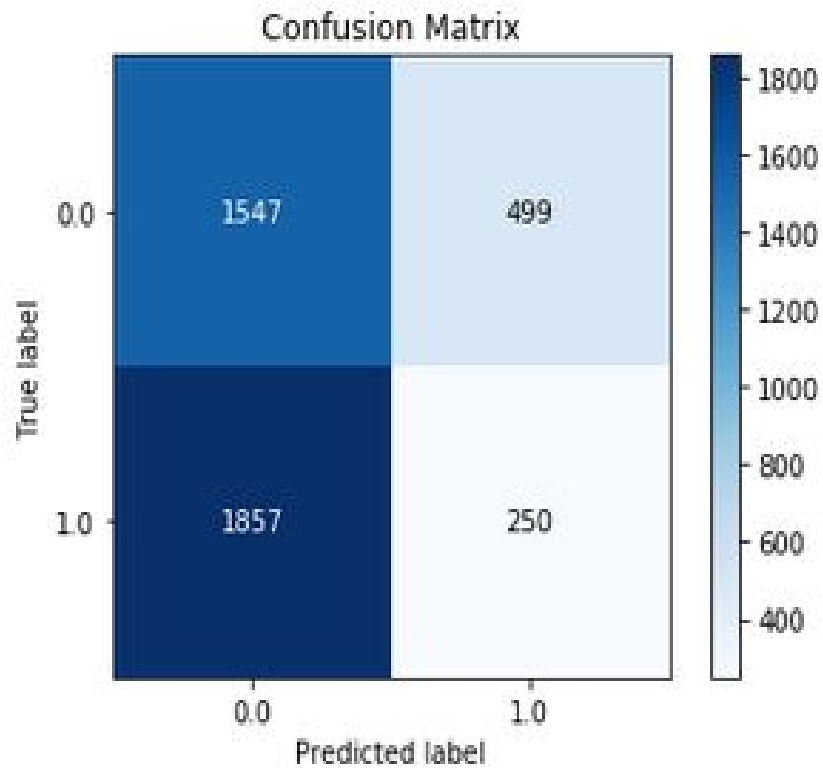


Fig 5.6: k-Means Clustering confusion matrix

**Accuracy obtained after implementation -**

**Accuracy: 40.37%**

## 5.4 EBPA

```
1 import numpy as np
2 xtr=np.load('E:\\Project\\xtr.npy');
3 xte=np.load('E:\\Project\\xte.npy');
4 ytr=np.load('E:\\Project\\ytr.npy');
5 yte=np.load('E:\\Project\\yte.npy');
6
7 from sklearn.linear_model import SGDClassifier as gd
8 import matplotlib.pyplot as plt
9
10 import scikitplot.plotters as skplt
11
12 clf=gd(loss='log',max_iter=100000,learning_rate='adaptive',eta0=0.00001)
13 clf.fit(xtr,ytr)
14 y_pred =clf.predict(xte)
15 m=yte.shape[0]
16 n=(yte != y_pred).sum()
17 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
18
19 skplt.plot_confusion_matrix(yte,y_pred)
20 plt.show()
```

Fig 5.7: EBPA implementation code

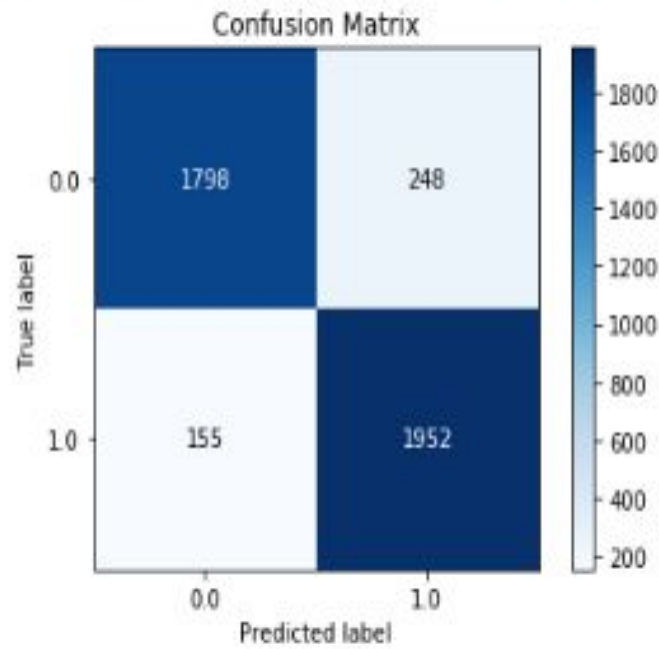


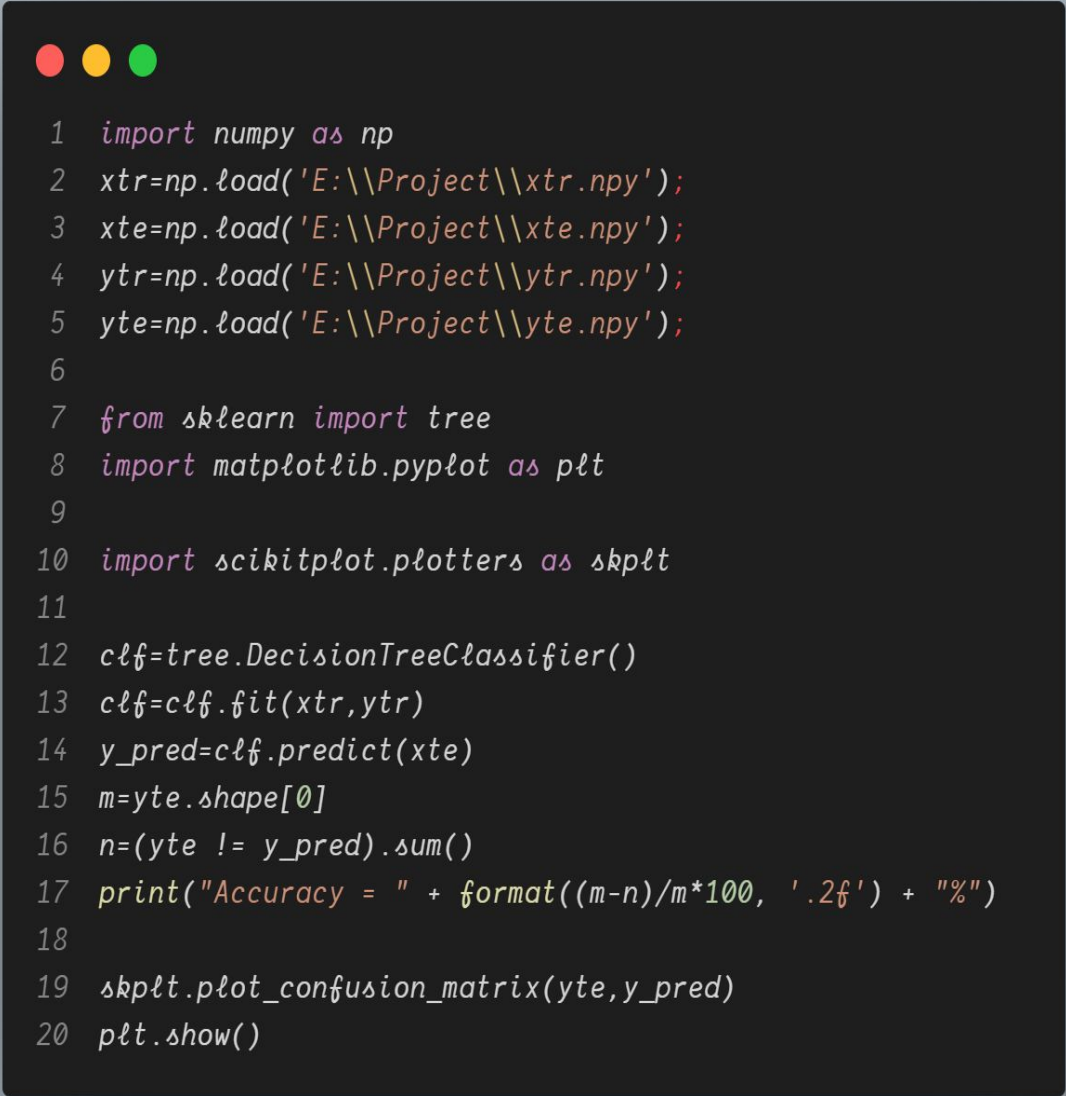
Fig 5.8: EBPA confusion matrix

**Accuracy obtained after implementation -**  
**Accuracy: 90.3%**

Table 5.1: EBPA with comparison of the training sets based on variation in iteration & learning rate parameters.

No. of Iterations	Constant eta	Adaptive eta
100	89.84	-
1000	90.32	90.42

## 5.5 Decision Tree



```
1 import numpy as np
2 xtr=np.load('E:\\Project\\xtr.npy');
3 xte=np.load('E:\\Project\\xte.npy');
4 ytr=np.load('E:\\Project\\ytr.npy');
5 yte=np.load('E:\\Project\\yte.npy');
6
7 from sklearn import tree
8 import matplotlib.pyplot as plt
9
10 import scikitplot.plotters as skplt
11
12 clf=tree.DecisionTreeClassifier()
13 clf=clf.fit(xtr,ytr)
14 y_pred=clf.predict(xte)
15 m=yte.shape[0]
16 n=(yte != y_pred).sum()
17 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
18
19 skplt.plot_confusion_matrix(yte,y_pred)
20 plt.show()
```

Fig 5.9: Decision Tree implementation code

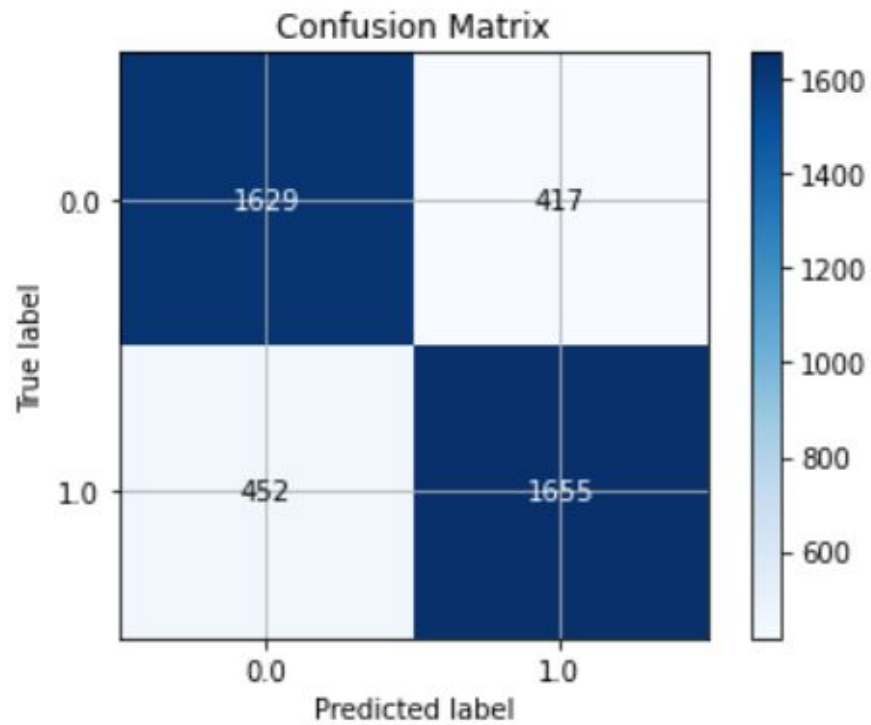
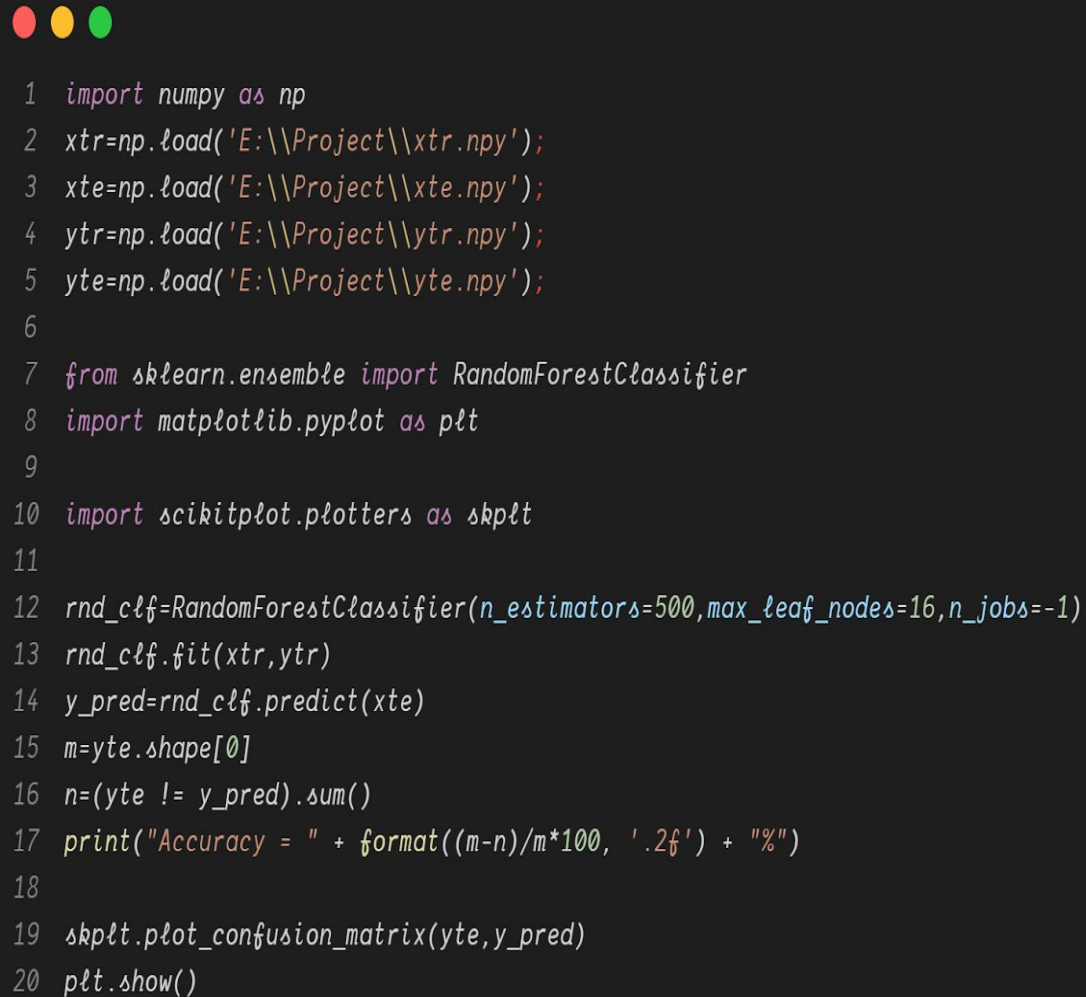


Fig 5.10: Decision Tree confusion matrix

**Accuracy obtained after implementation -**  
**Accuracy: 79.08%**



## 5.6 Random Forest



```
1 import numpy as np
2 xtr=np.load('E:\\Project\\xtr.npy');
3 xte=np.load('E:\\Project\\xte.npy');
4 ytr=np.load('E:\\Project\\ytr.npy');
5 yte=np.load('E:\\Project\\yte.npy');
6
7 from sklearn.ensemble import RandomForestClassifier
8 import matplotlib.pyplot as plt
9
10 import scikitplot.plotters as skplt
11
12 rnd_clf=RandomForestClassifier(n_estimators=500,max_leaf_nodes=16,n_jobs=-1)
13 rnd_clf.fit(xtr,ytr)
14 y_pred=rnd_clf.predict(xte)
15 m=yte.shape[0]
16 n=(yte != y_pred).sum()
17 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
18
19 skplt.plot_confusion_matrix(yte,y_pred)
20 plt.show()
```

Fig 5.11: Random Forests implementation code

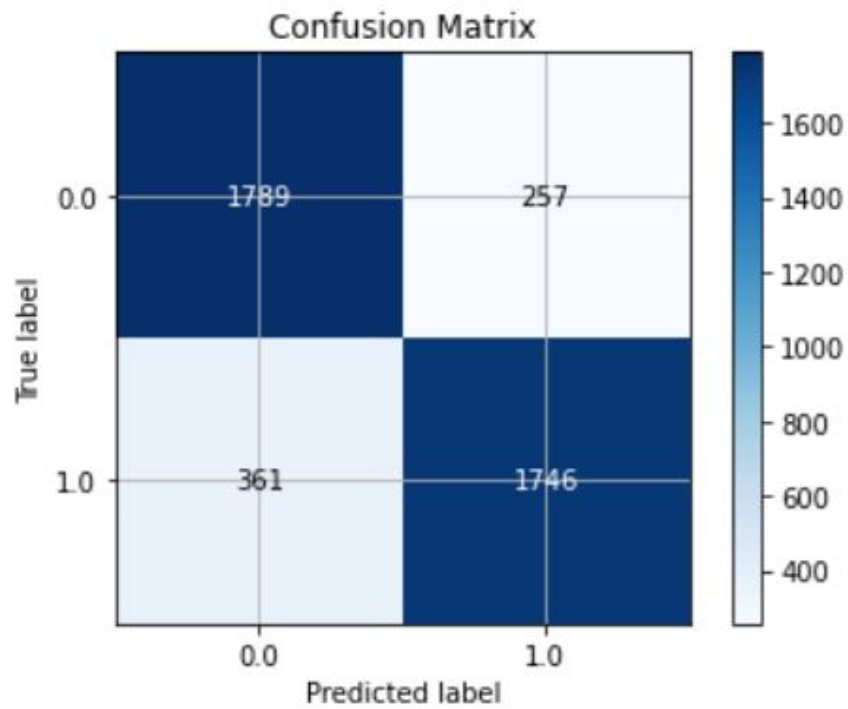
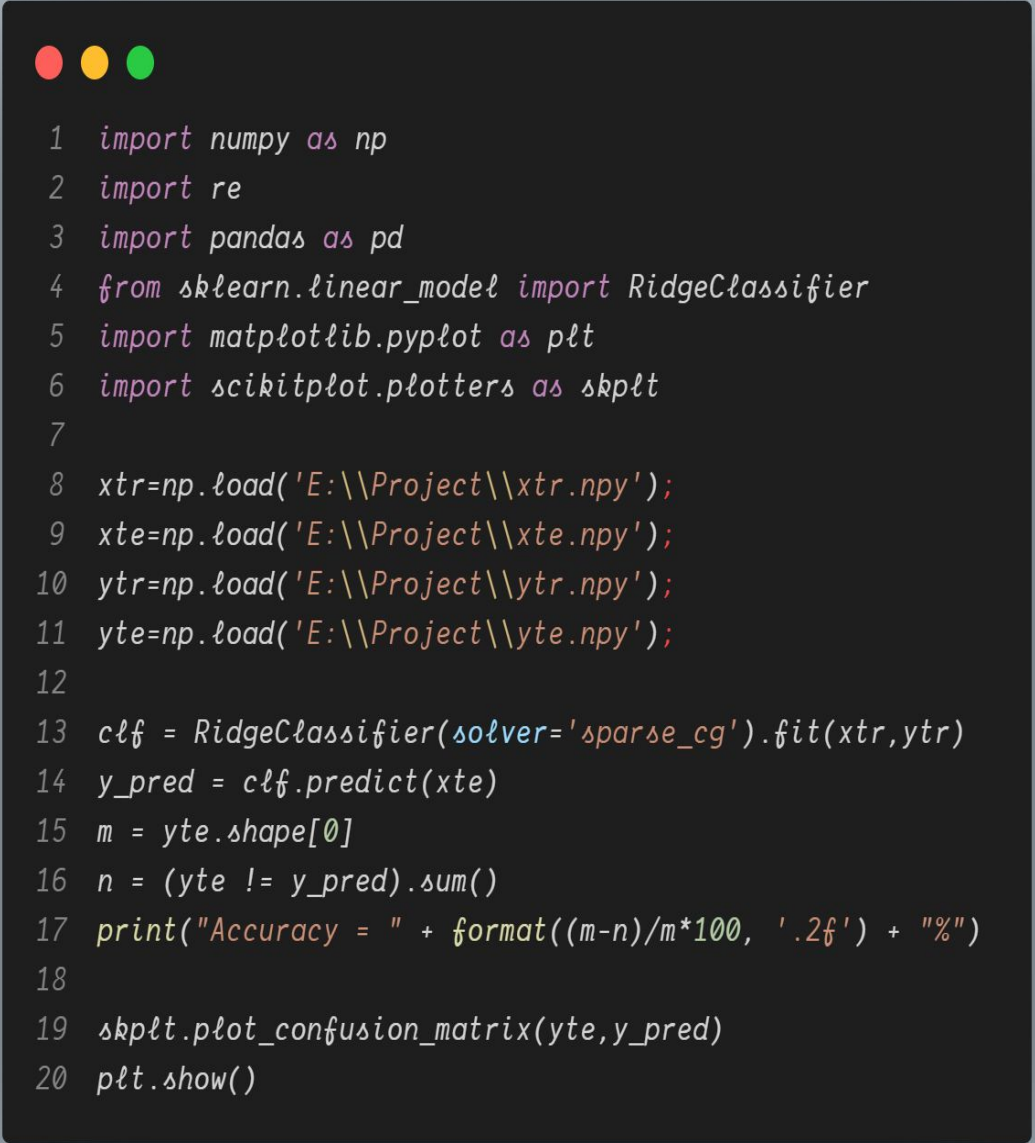


Fig 5.12: Random Forests confusion matrix

**Accuracy obtained after implementation -**  
**Accuracy: 85.12%**

## 5.7 Conjugate Gradient



```
1 import numpy as np
2 import re
3 import pandas as pd
4 from sklearn.linear_model import RidgeClassifier
5 import matplotlib.pyplot as plt
6 import scikitplot.plotters as skplt
7
8 xtr=np.load('E:\\Project\\xtr.npy');
9 xte=np.load('E:\\Project\\xte.npy');
10 ytr=np.load('E:\\Project\\ytr.npy');
11 yte=np.load('E:\\Project\\yte.npy');
12
13 clf = RidgeClassifier(solver='sparse_cg').fit(xtr,ytr)
14 y_pred = clf.predict(xte)
15 m = yte.shape[0]
16 n = (yte != y_pred).sum()
17 print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")
18
19 skplt.plot_confusion_matrix(yte,y_pred)
20 plt.show()
```

Fig 5.13: Conjugate Gradient implementation code

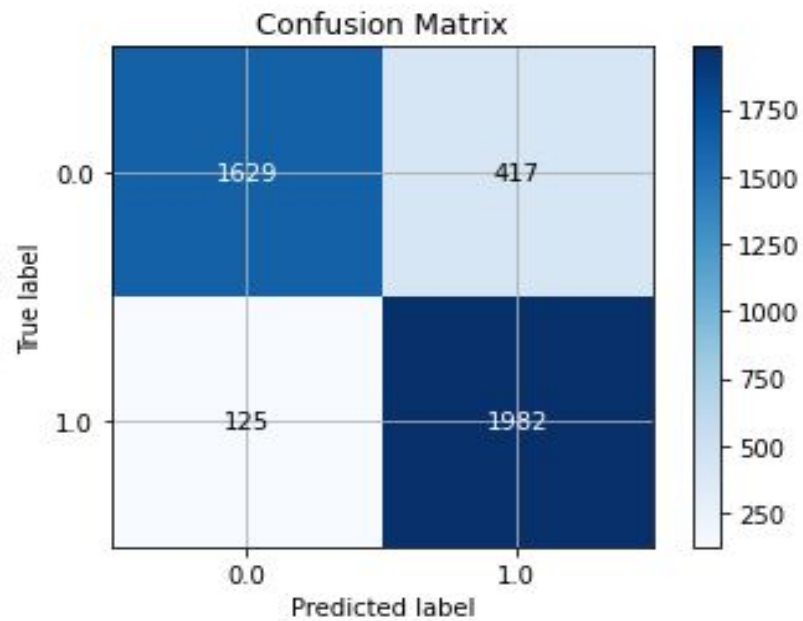


Fig 5.14: Conjugate Gradient confusion matrix

**Accuracy obtained after implementation -**  
**Accuracy: 86.95%**

# Chapter 6

## Results & Conclusion

---

Table 6.1: Various algorithms and their corresponding accuracy

Algorithm	Accuracy
Support Vector machines	88.47%
k-Nearest Neighbors	86.90%
k-Means Clustering	40.37%
Error back Propagation Algorithm	90.32% (constant eta) 90.42% (adaptive eta)
Decision Trees	79.08%
Random Forests	85.12%
Conjugate Gradient	86.95%

Error back propagation algorithm proved to be the best algorithm to predict fake news correctly with an adaptive type of learning behaviour. SVM and KNN also gave good results, if not best, with an accuracy of 88.47% and 86.90% respectively. While K-means was the worst and did not serve any purpose with a very low accuracy of fake news prediction.

# Chapter 7

## Future Scope

---

This project work aims to be utilised in various fields like checking criminal records, plagiarism.

- A best application would be checking criminal records(if any) of a particular person during investigation and research.
- Data plagiarism and patent registers: it can be helpful to identify and differentiate original content with those of its pirated copies.

# Bibliography

---

- [1] Nir Grinberg, Kenneth Joseph, Lisa Friedland, Briony Swire-Thompson, David Lazer, RESEARCH ARTICLE Fake news on Twitter during the 2016 U.S. presidential election, *Science* 25 Jan 2019:Vol. 363, Issue 6425, pp. 374-378 DOI: 10.1126/science.aau2706 Fake News.
- [2] Dataset reference: Kaggle <https://www.kaggle.com/c/fake-news/data>
- [3] Hieu Pham and Daniel Boley, An Empirical Approach to Sentiment Analysis with Doc2Vec, University of Minnesota, Computer Science and Engineering, doc2vec: <https://github.com/asprenger/doc2vec>.
- [4] Doc2vec documentation: [https://radimrehurek.com/gensim/auto\\_examples/tutorials/run\\_doc2vec\\_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py](https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py).
- [5] Theodoros Evgeniou and Massimiliano Pontil, WORKSHOP ON SUPPORT VECTOR MACHINES: THEORY AND APPLICATIONS, Center for Biological And Computational Learning, and Artificial Intelligence Laboratory, MIT, E25-201, Cambridge, MA 02139, USA.
- [6] DURGESH K. SRIVASTAVA, LEKHA BHAMBHU, DATA CLASSIFICATION USING SUPPORT VECTOR MACHINE, *Journal of Theoretical and Applied Information Technology*.
- [7] Oyelade, O. J., Oladipupo, O. O., Obagbuwa, I. C., Application of k-Means Clustering algorithm for prediction of Students' Academic Performance, (IJCSIS) *International Journal of Computer Science and Information Security*, Vol. 7, o. 1, 2010.
- [8] Imad Dabbura (Sep 17, 2018) K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks, Towards datascience\_ref: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>.
- [9] Yun-lei Cai, Duo Ji, Dong-feng Cai, A KNN Research Paper Classification Method Based on Shared Nearest Neighbor, *Proceedings of NTCIR-8 Workshop Meeting*, June 15–18, 2010, Tokyo, Japan.
- [10] Sebastian Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747v2 [cs.LG] 15 Jun 2017.
- [11] <https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html>
- [12] <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
- [13] [https://en.wikipedia.org/wiki/Bootstrapping\\_\(statistics\)](https://en.wikipedia.org/wiki/Bootstrapping_(statistics))
- [14] [https://www.researchgate.net/figure/An-example-of-bootstrap-sampling-Since-objects-are-subsampled-with-replacement-some\\_fig2\\_322179244](https://www.researchgate.net/figure/An-example-of-bootstrap-sampling-Since-objects-are-subsampled-with-replacement-some_fig2_322179244)
- [15] Fernando Cardoso Durier da Silva, Rafael Vieira da Costa Alves, Ana Cristina Bicharra Garcia, Can Machines Learn to Detect Fake News? A Survey Focused on Social Media, *Proceedings of the 52nd Hawaii International Conference on System Sciences* | 2019n URI: <https://hdl.handle.net/10125/59713> ISBN: 978-0-9981331-2-6 (CC BY-NC-ND 4.0) Page 2763.