Term Deposit Subscription Prediction

DEALING WITH DATA

By

MOHIT ADHIKARI (MGFM-1901)

ADITYA BAKDE (MFT-1902)

DIVYAM BHATT (MFT-1903)

ROHIT RAMASWAMY (MFT-1912)

UNDER THE GUIDANCE OF

Prof. Pranav Shastri

BSE INSTITUTE LTD.
2019 – 2021

Abstract

The objective of this project will be the prediction of whether the client will subscribe to a term deposit or not. For its training and testing we have used a dataset consisting of user's data. This data is related with direct marketing campaigns of a Portuguese banking institution collected by phone calls. The data consists of the bank client data, data pertaining to social and economic attributes. To implement this multiple algorithms have been used and the accuracy of each algorithm is compared to see which approach is most suitable for this case.

Keywords: *Machine Learning, Categorical Encoding, Decision tree, Random Forest, Naive Bayes, Clustering.*

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Technology used

1.1.1 Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python is used for:

1) Web Development(Server-side)
2) Software Development
3) Mathematics
4) System Scripting
5) Data Analytics

What Python can do:

- Python can be used on a server to create web applications
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

IDE for Python

An IDE (or Integrated Development Environment) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development. These tools usually include:

- An editor designed to handle code (with, for example, syntax highlighting and auto-completion)
- Build, execution, and debugging tools
- Some form of source control

Most IDEs support many different programming languages and contain many more features. They can, therefore, be large and take time to download and install. You may also need advanced knowledge to use them properly.

In contrast, a dedicated code editor can be as simple as a text editor with syntax highlighting and code formatting capabilities. Most good code editors can execute code and control a debugger. The very best ones interact with source control systems as well. Compared to an IDE, a good dedicated code editor is usually smaller and quicker, but often less feature rich.

For the past couple of years, Jupyter Notebook has been gaining a lot of popularity in terms of coding and debugging. Notebooks have been redefining the concept of an IDE and are adding more and more features into it.

Let's see which are the most popular Python IDE and text editors. It's important to note that the below statistics are from December 2018 and should give a good understanding of what is popular today.

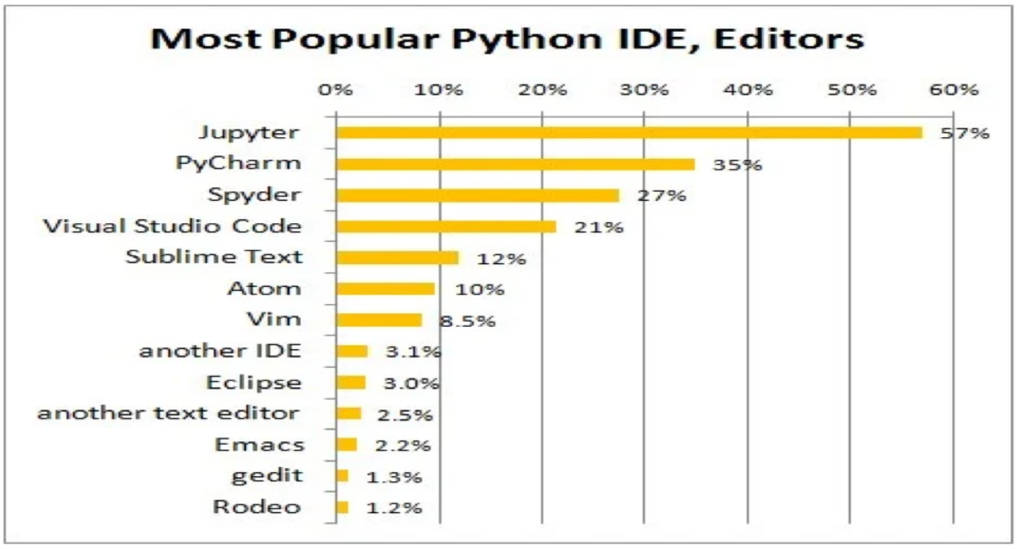


Figure 1.1: Python IDE Comparison

1.1.2 Tableau

Tableau is a powerful and fastest growing data visualization tool used in the Business Intelligence Industry. It helps in simplifying raw data into the very easily understandable format.

Data analysis is very fast with Tableau and the visualizations created are in the form of dashboards and worksheets. The data that is created using Tableau can be understood by professional at any level in an organization. It even allows a non-technical user to create a customized dashboard.

The best features of Tableau are:

- Data Blending
- Real-time analysis
- Collaboration of Data

The great thing about Tableau software is that it doesn't require any technical or any kind of programming skills to operate. The tool has garnered interest among the people from all sectors such as business, researchers, different industries, etc.

What is tableau used for?

- Tableau software is used to translate queries into visualization.
- It is also used for managing metadata.
- Tableau software imports data of all sizes and ranges.
- For a non-technical user, Tableau is a life saver as it offers the facility to create 'no-code' data queries

Ever since it was introduced, this data visualization tool is used for Business Intelligence industry. Organizations like Amazon, Walmart, Accenture, Lenovo, and so on widely use Tableau.

1.2 Inspiration

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages modular programs and code reuse ability. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast.

Tableau

Tableau is greatly used because data can be analyzed very quickly with it. Also, visualizations are generated as dashboards and worksheets. Tableau allows one to create dashboards that provide actionable insights and drives the business forward. Tableau products always operate in virtualised environments when they are configured with the proper underlying operating system and hardware. Tableau is used to explore data with limitless visual analytics.

Chapter 2

Topic and Dataset

2.1 Topic

The objective of this project will be the prediction of whether the client will subscribe to a term deposit or not based on the dataset observations using various algorithms and calculating their accuracy.

2.2 Dataset

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The dataset includes bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analysed in [Moro et al., 2014]. The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

Input variables:

Bank client data:

1) age (numeric)
2) job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid', 'management', 'retired','self- employed','services','student', 'technician', 'unemployed','unknown')
3) marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4) education (categorical: ’basic.4y’,’basic.6y’,’basic.9y’,’high.school’,’illiterate’, ’professional.course’, ’university.degree’,’unknown’)

5) default: has credit in default? (categorical: ’no’,’yes’,’unknown’)

6) housing: has housing loan? (categorical: ’no’,’yes’,’unknown’)

7) loan: has personal loan? (categorical: ’no’,’yes’,’unknown’)

Related with the last contact of the current campaign:

1) contact: contact communication type (categorical: ’cellular’,’telephone’)

2) month: last contact month of year (categorical: ’jan’, ’feb’, ’mar’, ..., ’nov’, ’dec’)

3) day_of_week: last contact day of the week (categorical: ’mon’,’tue’,’wed’,’thu’,’fri’)

4) duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y=’no’). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes

1) campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

2) pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

3) previous: number of contacts performed before this campaign and for this client (numeric)

4) poutcome: outcome of the previous marketing campaign (categorical: ’failure’, ’nonexistent’,’success’)

Output variable (desired target)

1) y - has the client subscribed a term deposit? (binary: ’yes’,’no’)ic/data set selected (different traits of dataset)

Chapter 3

Implementation

3.1 Data Pre-processing

Pre-processing is the cleaning of the dataset before it is sent as input for the algorithms, the basic pre-processing will be done for all the algorithms performed and the subsequent data will be stored in a new file for better understanding. As of now no preparations have been made for algorithm specific pre-processing as it will lead to unfair comparisons of the accuracies of the algorithms. For elementary pre-processing the missing values will be eliminated, the extreme outliers shall be trimmed, as far as the attributes are concerned the team has decided to go with trial and error method for selection of the same, the attributes providing highest accuracy will be shortlisted while the others shall be discarded. The project will be a derived on the idea of algorithmic comparisons so as to learn about selection of algorithms on the type of data analysed. In case the categorical attributes are selected, the categories shall be segregated in individual columns for faster implementation and for better understanding. Before pre-processing, the summary of the attribute will be noted for planning of pre-processing operations that shall be done, and also for comparing with processed data.

3.1.1 Categorical Encoding

Machine Learning models are based on numeric equations and calculation of numeric variables. At times we do have columns in out dataset which are non-numeric. These are categorical variables (eg. Country, Size, Occupation etc.).

In such conditions we need to convert these values into numeric values to help us in computation and further processing this data. There are multiple ways to deal with this problem. The two libraries Pandas and scikit-learn make this job easy.

The simplest way to achieve categorical encoding in the dataset for categorical variables is to assign a number to each distinct category within the list of attributes(column). In order to achieve this, the LabelEncoder class of the scikit-learn package is used. This is shown below with the help of an example.

Lets consider this example dataset shown below:

Country	**Hours**	**Salary**
France	34	12000
Spain	23	10000
Germany	20	15000
Spain	30	12000
France	28	14000
France	24	9000
Germany	32	19000
Spain	26	10000

Table 3.1: Categorical Encoding Example Dataset

Here we need to encode the Country column as that is the only categorical variable present in the dataset, the remaining are all numeric values and are ready for processing. We use LabelEncoder class to do the same. This would assign a value to each country and replace the name of the country by its corresponding value.

```python
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0]) # All rows and first column
     i.e country column
print(X)
```

Listing 3.1: LabelEncoder

The result is now:

```
array([[ 0, 34, 12000],
       [ 1, 23, 10000],
       [ 2, 20, 15000],
       [ 1, 30, 12000],
       [ 0, 28, 14000],
       [ 0, 24, 9000],
       [ 2, 32, 19000],
       [ 1, 26, 10000], dtype=object)
```

Listing 3.2: LabelEncoder output

The column Country has now been encoded. We can see that the three countries have been given a unique number starting from 0 going up to 2 (0,1,2). However there is a problem with this method. We cannot just find and replace the categorical values by a distinct numeric value. The machine learning algorithms would now treat this as numbers, and hence consider class 2 to be greater than class 1 which is not really the case. These numbers are denoting countries and countries cannot be ranked in relative order. This would hence think that Germany is greater than Spain, which should not be the case. This could be applied when we take into consideration a category which has values that can be relatively ranked. For example a category 'size' can have values 'Small', 'Medium', and 'Large'. This can be encoded as 0,1 and 2 respectively.

Now even if the model sees this mathematically and infers that Small is less than Medium since 0 is less than 1, it stands to be true because the values can be ranked relatively.

In order to prevent this problem of the model inferring relative ranking, we use OneHotEncoder. This introduces dummy variables(columns). This converts the single column of countries into n number of columns, n being the number of distinct countries in the list.

```
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
print(X)
```

Listing 3.3: OneHotEncoder

The result can be seen as:

```
Country   France  Spain  Germany
France    1       0      0
Spain     0       1      0
Germany   0       0      1
Spain     0       1      0
France    1       0      0
France    1       0      0
Germany   0       0      1
Spain     0       1      0
```

Listing 3.4: OneHotEncoder output

Even though this is a good approach and eliminates the previous problems, in a dataset having large number of columns already this will add more columns. Moreover this will add multiple columns, each column for each distinct value in a column. This would add many columns and make the already big dataset into complex ones.

The best approach is to factorize the columns having categorical values. This does not add additional columns and replaces categorical values with numeric ones. Here the numeric values are treated as factors, an enumerated list and eliminate the possibility of relative ranking.

The implementation of factorize in our project is shown in the snippet below. The two columns 'job' and 'marital' have been factorized to achieve categorical encoding.

```
processed_encoding['job']=pd.factorize(processed_data.job)[0]
processed_encoding['marital']=pd.factorize(processed_data.marital)[0]
```

Listing 3.5: Factorize implementation

3.1.2 Splitting Dataset

In machine learning, usually our data is split into two subsets: training data and testing data (and sometimes to three: train, validate and test), and our model is fitted on the train data, in order to make predictions on the test data. On doing this, one of two things might happen: our model is over fitted or under fitted. Neither of these are desirable, because they affect the predictability of the model — the model might have lower accuracy and/or is not generalized (meaning you can't generalize your predictions on other data). Let's see what under and over fitting actually mean:

Over fitting

Over fitting means that the model trained has trained "too well" and is now fit too closely to the training dataset. This usually happens when the model is too complex (i.e. too many features/variables compared to the number of observations). This model will be very accurate on the training data but will probably be very not accurate on untrained or new data. It is because this model is not generalized (or not AS generalized), meaning you can generalize the results and can't make any inferences on other data, which is, ultimately, what you are trying to do. Basically, when this happens, the model learns or describes the "noise" in the training data instead of the actual relationships between variables in the data. This noise, obviously, isn't part in of any new dataset, and cannot be applied to it.

Under fitting

In contrast to over fitting, when a model is under fitted, it means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data. This is usually the result of a very simple model (not enough predictors/independent variables). It could also happen when, for example, we fit a linear model (like linear regression) to data that is not linear. It almost goes without saying that this model will have poor predictive ability (on training data and can't be generalized to other data).

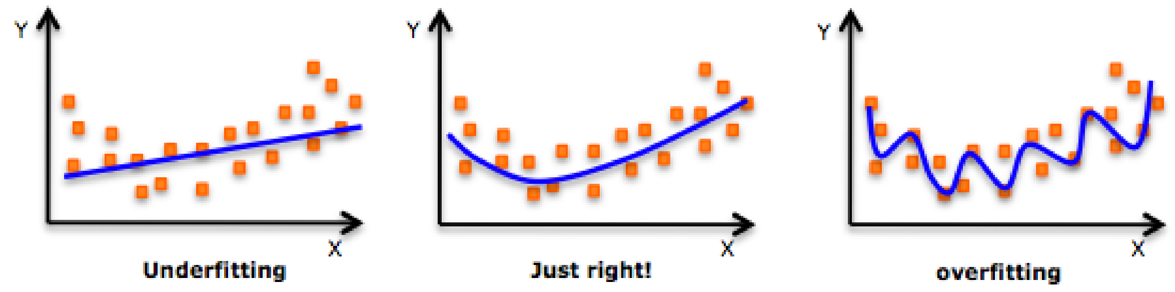


Figure 3.1: Example of over fitting and under fitting

It is worth noting the under fitting is not as prevalent as over fitting. Nevertheless, we want to avoid both of those problems in data analysis. We are trying to find the middle ground between under and over fitting the model.

Train-Test Split

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model's prediction on this subset.

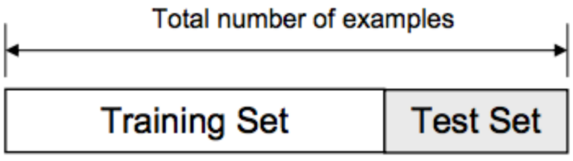


Figure 3.2: Train test split

```python
from sklearn.model_selection import train_test_split

X = processed_encoding.iloc[1:, 0:15]
Y = processed_encoding.iloc[1:, -1]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
    random_state=1)
```

Listing 3.6: Train Test Split implementation

The complete dataset has hence been divided into two sets: train set and the testing set. This has been done for both, the set of independent variables and well as a set of the dependent variable. This hence results in four sets comprising of the entire data. There is no loss of data while splitting. 70% of the data has been taken for training and the remaining 30% has been kept for testing.

3.2 Algorithms

3.2.1 Decision tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too.

The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data(training data).

The understanding level of Decision Trees algorithm is so easy compared with other classification algorithms. The decision tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label.

Decision Tree Algorithm Pseudo code:

1) Place the best attribute of the dataset at the root of the tree.

2) Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.

3) Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

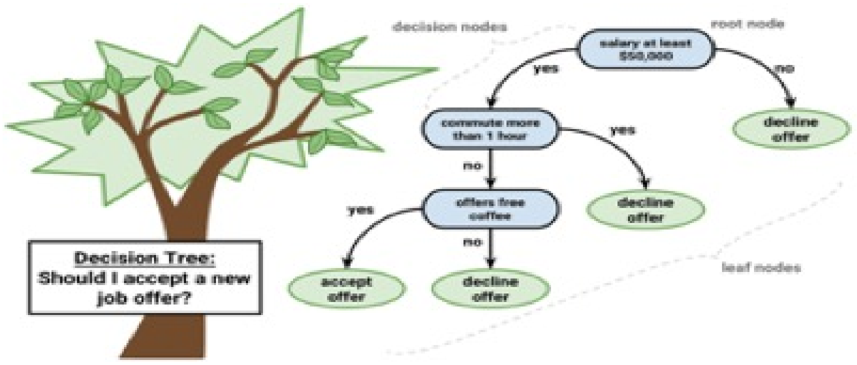


Figure 3.3: Decision Tree Pseudo code Example

In decision trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

We continue comparing our record's attribute values with other internal nodes of the tree until we reach a leaf node with predicted class value. As we know how the modeled decision tree can be used to predict the target class or the value. Now let's understanding how we can create the decision tree model.

Assumptions while creating Decision Tree

- At the beginning, the whole training set is considered as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

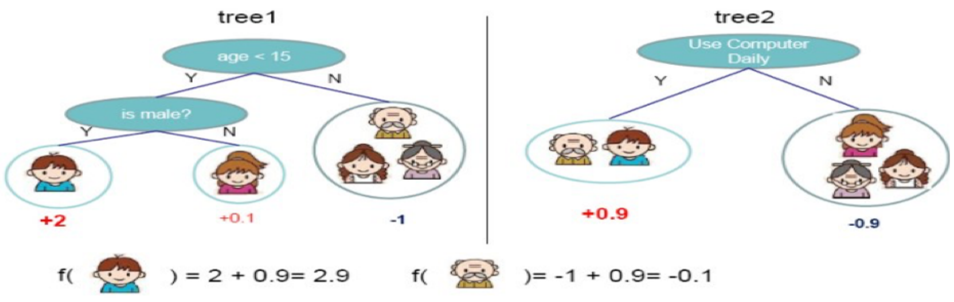


Figure 3.4: Decision Tree Assumptions Illustration

Decision Trees follow Sum of Product (SOP) representation. For the above images, you can see how we can predict can we accept the new job offer? and Use computer daily? from traversing for the root node to the leaf node.

It's a sum of product representation. The Sum of product(SOP) is also known as Disjunctive Normal Form. For a class, every branch from the root of the tree to a leaf node having the same class is a conjunction(product) of values, different branches ending in that class form a dis junction(sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is know the attributes selection. We have different attributes selection measure to identify the attribute which can be considered as the root note at each level.

Attribute Selection

If dataset consists of "n" attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can't solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some criterion like information gain, gini index, etc. These criterions will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e, the attribute with a high value(in case of information gain) is placed at the root.

While using information Gain as a criterion, we assume attributes to be categorical, and for gini index, attributes are assumed to be continuous.

Over fitting

Over fitting is a practical problem while building a decision tree model. The model is having an issue of over fitting is considered when the algorithm continues to go deeper and deeper in the to reduce the training set error but results with an increased test set error i.e, Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data.

Two approaches to avoid over fitting:

- Pre-Pruning
- Post-Pruning

Pre-Pruning

In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it's difficult to choose an appropriate stopping point.

Post-Pruning

In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the over fitting problem then pruning is done as a post-pruning step. We use a cross-validation data to check the effect of our pruning. Using cross-validation data, it tests whether expanding a node will make an improvement or not.

If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded i.e, the node should be converted to a leaf node.

Advantages

1) Decision Trees are easy to explain. It results in a set of rules.
2) It follows the same approach as humans generally follow while making decisions.
3) Interpretation of a complex Decision Tree model can be simplified by its visualizations. Even a naive person can understand logic.
4) The Number of hyper-parameters to be tuned is almost null.

Disadvantages

1) There is a high probability of over fitting in Decision Tree.
2) Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
3) Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.
4) Calculations can become complex when there are many class labels.

3.2.2 Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).

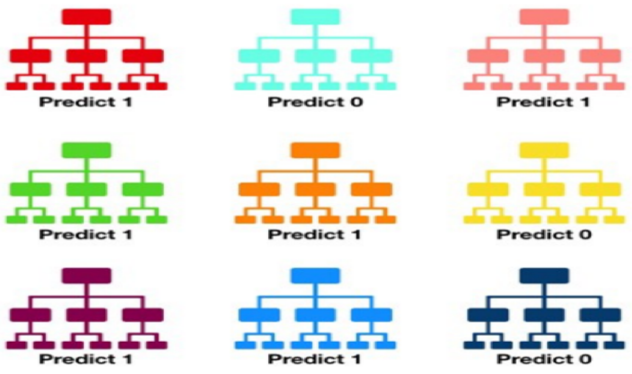


Figure 3.5: Random Forest

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is, a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

We can understand the working of Random Forest algorithm with the help of following steps

1) Start with the selection of random samples from a given dataset.
2) Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
3) In this step, voting will be performed for every predicted result.
4) At last, select the most voted prediction result as the final prediction result.

Random Forest Real Life Example

Suppose Mady wants to go to different places that he may like for his two-week vacation, and he asks his friend for advice. His friend will ask where he has been to already, and whether he likes the places that he's visited. Based on Mady's answers, his friend starts to give the recommendation. Here, his friend forms the decision tree.

Mady wants to ask more friends for advice because he thinks only one friend cannot help him make an accurate decision. So his other friends also ask him random questions, and finally, provides an answer. He considers the place with the most votes as his vacation decision. Here, the author provides an analysis for this example.

His one friend asked him some questions and gave the recommendation of the best place based on the answers. This is a typical decision tree algorithm approach. The friend created the rules based on the answers and used the rules to find the answer that matched the rules.

Mady's friends also randomly asked him different questions and gave answers, which for Mady are the votes for the place. At the end, the place with the highest votes is the one Mady will select to go. This is the typical Random Forest algorithm approach.

Advantages

- It overcomes the problem of over fitting by averaging or combining the results of different decision trees.
- Random forests work well for a large range of data items than a single decision tree does.
- Random forest has less variance then single decision tree.
- Random forests are very flexible and possess very high accuracy.
- Scaling of data does not require in random forest algorithm. It maintains good accuracy even after providing data without scaling.
- Random Forest algorithms maintains good accuracy even a large proportion of the data is missing.

Disadvantages

- Complexity is the main disadvantage of Random forest algorithms.
- Construction of Random forests are much harder and time-consuming than decision trees.
- More computational resources are required to implement Random Forest algorithm.
- It is less intuitive in case when we have a large collection of decision trees.
- The prediction process using random forests is very time-consuming in comparison with other algorithms.

3.2.3 Naive Bayes

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

What is Naive Bayes Theorem?
Bayes theorem named after Rev. Thomas Bayes. It works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. Using the conditional probability, we can calculate the probability of an event using its prior knowledge.

Below is the formula for calculating the conditional probability:

$$P(H/E) = \frac{P(E/H) * P(H)}{P(E)}$$

Where,

- P(H) is the probability of hypothesis H being true. This is known as the prior probability.
- P(E) is the probability of the evidence(regardless of the hypothesis).
- P(E|H) is the probability of the evidence given that hypothesis is true.
- P(H|E) is the probability of the hypothesis given that the evidence is there.

Take a simple machine learning problem, where we need to learn our model from a given set of attributes(in training examples) and then form a hypothesis or a relation to a response variable. Then we use this relation to predict a response, given attributes of a new instance. Using the Bayes' theorem, its possible to build a learner that predicts the probability of the response variable belonging to some class, given a new set of attributes.

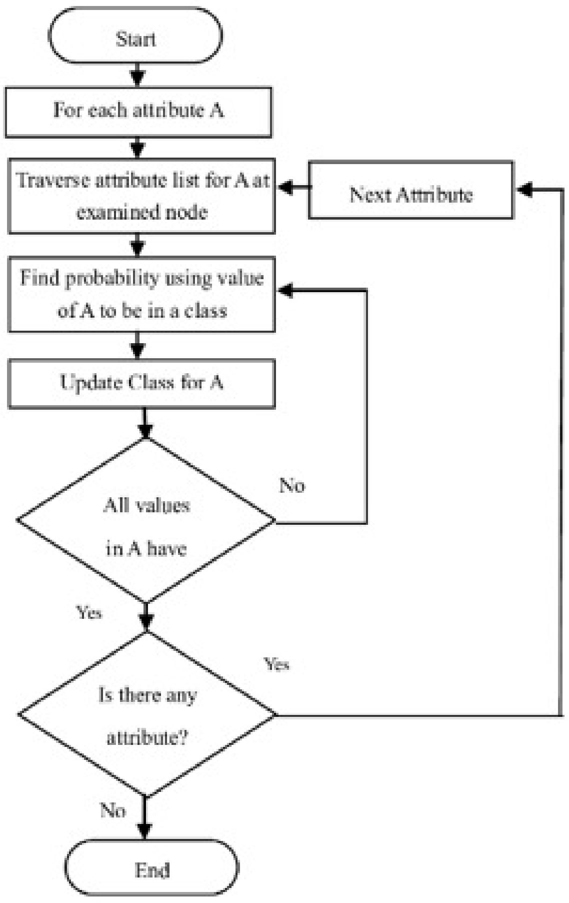
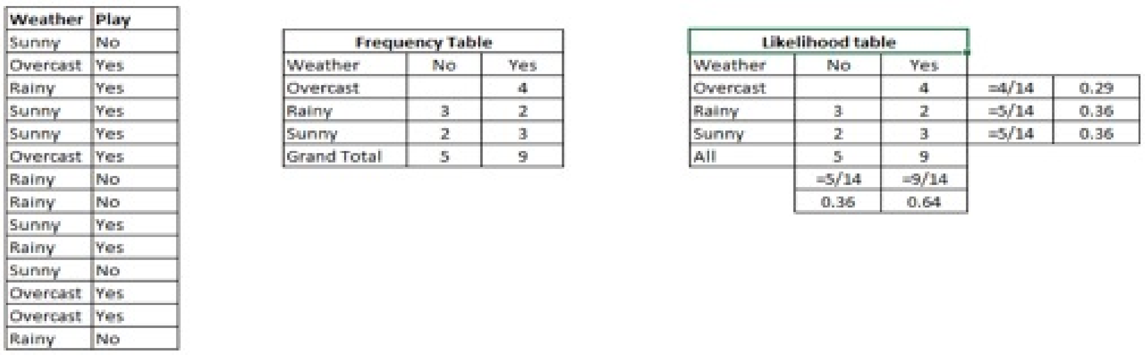


Figure 3.6: Naive Bayes Flowchart

Example: Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play'. Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Steps:

1) Convert the data set to frequency table.
2) Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.
3) Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.



Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Figure 3.7: Naive Bayes Steps

Problem: Players will pay if weather is sunny, is this statement is correct?
We can solve it using above discussed method, so

$$P(Yes/Sunny) = \frac{P(Sunny/Yes) * P(Yes)}{P(Sunny)}$$

Here we have

$$P(Sunny/Yes) = \frac{3}{9} = 0.33$$

$$P(Sunny) = \frac{5}{14} = 0.36$$

$$P(Yes) = \frac{9}{14} = 0.64$$

Now,

$$P(Yes/Sunny) = \frac{0.33 * 0.64}{0.36} = 0.60$$

has a higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Types of Naive Bayes Algorithms

Gaussian Naive Bayes It is used in classification and it assumes that features follow a normal distribution.

Multi Nomial Naive Bayes Multi Nomial Naive Bayes is preferred to use on data that is multinomially distributed. It is one of the standard classic algorithms. Which is used in text categorization (classification). Each event in text classification represents the occurrence of a word in a document.

Bernaulli Naive Bayes Bernoulli Naive Bayes is used on the data that is distributed according to multivariate Bernoulli distributions.i.e., multiple features can be there, but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. So, it requires features to be binary valued.

Advantages:

- Naive Bayes Algorithm is a fast, highly scalable algorithm.
- Naive Bayes can be use for Binary and Multi-class classification. It provides different types of Naive Bayes Algorithms like GaussianNB, MultinomialNB, BernoulliNB.
- It is a simple algorithm that depends on doing a bunch of counts.
- Great choice for Text Classification problems. It's a popular choice for spam email classification.
- It can be easily train on small dataset.

Disadvantages:

It considers all the features to be unrelated, so it cannot learn the relationship between features. E.g., Let's say Remo is going to a part. While cloth selection for the party, Remo is looking at his cupboard. Remo likes to wear a white color shirt. In Jeans, he likes to wear a brown Jeans, But Remo doesn't like wearing a white shirt with Brown Jeans. Naive Bayes can learn individual features importance but can't determine the relationship among features.

3.2.4 K-Means

K-means is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

Being a clustering algorithm, k-Means takes data points as input and groups them into k clusters. This process of grouping is the training phase of the learning algorithm. The result would be a model that takes a data sample as input and returns the cluster that the new data point belongs to, according the training that the model went through. How can this be useful? Well, that's how content promotion and recommendation usually works — in a very simplistic manner. Websites may choose to put people in bubbles (i.e. clusters) with other people who share similar activities (i.e. features) on the website. This way, the recommended content will be somewhat on-point, as existing users with similar activities are likely to be interested in similar content. Moreover, as a new person goes into the ecosystem of the website, that person will be placed within a particular cluster, and the content recommendation system takes care of the rest. Building on that idea, k-Means is just a clustering algorithm. It uses the distance between points as a measure of similarity, based on k averages (i.e. means).

The idea behind k-Means is that, we want to add k new points to the data we have. Each one of those points — called a Centroid — will be going around trying to center itself in the middle of one of the k clusters we have. Once those points stop moving, our clustering algorithm stops.

As you might've suspected, the value of k is of great importance. This k is called a hyper-parameter; a variable whose value we set before training. This k specifies the number of clusters we want the algorithm to yield. This number of clusters is actually the number of centroids going around in the data.

Working

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids.

It halts creating and optimizing clusters when either:

1) The centroids have stabilized — there is no change in their values because the clustering has been successful.
2) The defined number of iterations has been achieved.

Steps:

1) Choose a value of k, number of clusters to be formed.
2) Randomly select k data points from the data set as the intital cluster centeroids/centers.
3) For each datapoint:
    - Compute the distance between the datapoint and the cluster centroid.
    - Assign the datapoint to the closest centroid.
4) For each cluster calculate the new mean based on the datapoints in the cluster.
5) Repeat 3 & 4 steps until mean of the clusters stops changing or maximum number of iterations reached.

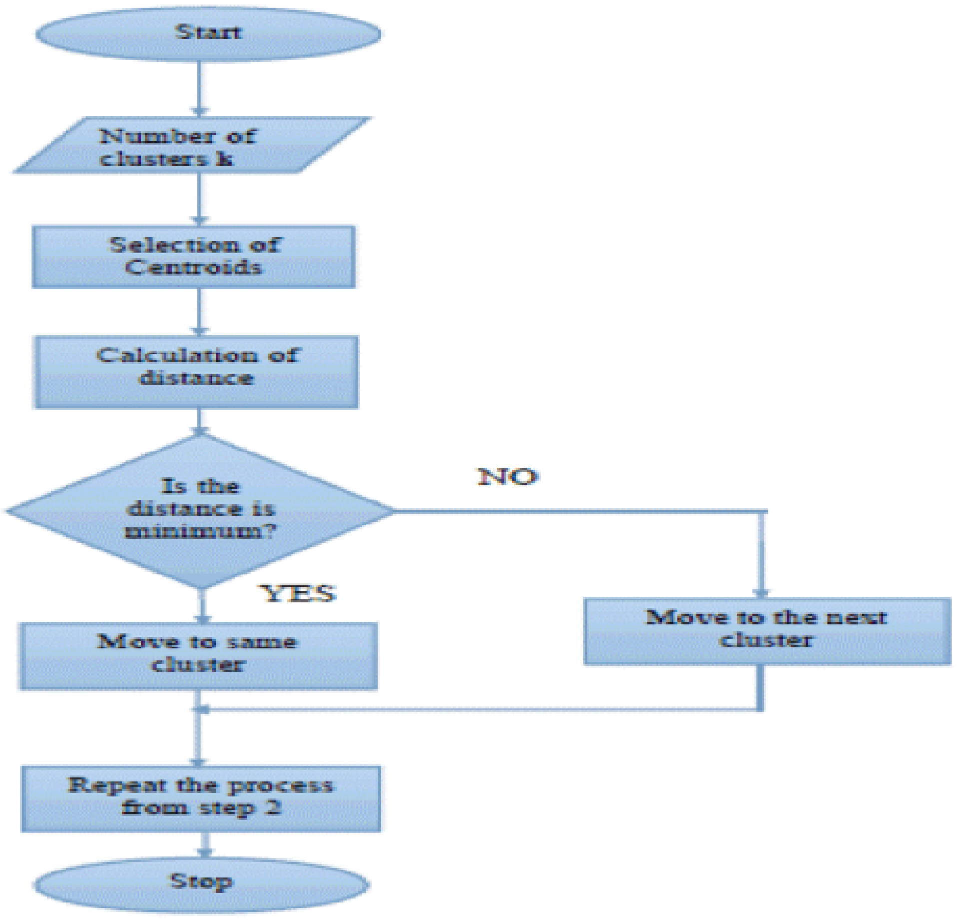


Figure 3.8: K-Means working

Understanding with an example

We will apply k-means on the following 1 dimensional data set for K=2.

Data set 2, 4, 10, 12, 3, 20, 30, 11, 25

Iteration 1:

M1, M2 are the two randomly selected centroids/means where: M1= 4, M2=11. The initial clusters are: C1= 4, C2= 11. Calculate the euclidean distance as: $D[x, a] = \sqrt{(x-a)^2}$

Where, D1 is the distance from M1 & D2 is the distance from M2.

Datapoint	**D1**	**D2**	**Cluster**
2	2	9	C1
4	0	7	C1
10	6	1	C2
12	8	1	C2
3	1	8	C1
20	16	9	C2
30	26	19	C2
11	7	0	C2
25	21	14	C2

Table 3.2: K-Means Iteration 1

As we can see in Table 3.2, 3 datapoints are added to cluster C1 and the other datapoints are added in cluster C2. Therefore, C1 = {2,4,3} and C2 = {10,12,20,30,11,25}.

Iteration 2: Calculate new mean of datapoints in C1 and C2. $M1 = \frac{(2+3+4)}{3} = 3$

$M2 = \frac{(10+11+12+20+25+30)}{6} = 18$
Calculating distance and updating clusters based on table below

Datapoint	**D1**	**D2**	**Cluster**
2	1	16	C1
4	1	14	C1
3	0	15	C1
10	7	8	C1
12	9	6	C2
20	17	2	C2
30	27	12	C2
11	8	7	C2
25	22	7	C2

Table 3.3: K-Means Iteration 2

New clusters: C1 = 2,3,4,10 and C2 = 11,12,20,25,30

Iteration 3 Calculate new mean of datapoints in C1 and C2:
M1= 4.75 & M2 = 19.6 Calculating distance and updating clusters based on table below:

Datapoint	D1	D2	Cluster
2	2.75	17.6	C1
4	0.75	15.6	C1
3	1.75	16.6	C1
10	5.25	9.6	C1
12	7.25	7.6	C2
20	15.25	0.4	C2
30	25.25	10.4	C2
11	6.25	8.6	C1
25	20.25	5.4	C2

Table 3.4: K-Means Iteration 3

New clusters: C1 = 2,3,4,10,11,12 and C2 = 20,25,30

Iteration 4: Calculate new mean of datapoints in C1 and C2.
M1 = 7 & M2 = 25. Calculating distance and updating clusters based on table below:

Datapoint	D1	D2	Cluster
2	5	23	C1
4	3	21	C1
3	4	22	C1
10	3	15	C1
12	5	13	C1
11	4	14	C1
20	13	5	C2
30	23	5	C2
25	18	0	C2

Table 3.5: K-Means Iteration 4

New Clusters are C1= 2, 3, 4, 10, 12, 11 & C2= 20, 30, 25.
As we can see that the data points in the cluster C1 and C2 in iteration 3 are same as the data points of the cluster C1 and C2 of iteration 4. It means that none of the data points has moved to other cluster. Also the means/centeroid of these clusters is constant. So this becomes the stopping condition for our algorithm.

Chapter 4

Results

Multiple algorithms were implemented in order to achieve the objective of the project. Along with three prediction algorithms there was one clustering algorithm also used. The clustering algorithm was used to visualise the entire data points in clusters. The prediction algorithms were trained on the training test which had 70% of the data and were tested against the testing set which comprised of 30% of the dataset. The results of each of the algorithms along with their respective accuracy are discussed below:

4.1 Naive Bayes

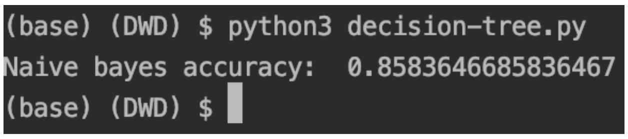
Implementation

```python
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

def naive_bayes():
  model = GaussianNB()
  model.fit(X_train,Y_train)
  Y_pred = model.predict(X_test)
  print("Naive bayes accuracy: ", metrics.accuracy_score(Y_test, Y_pred))
```

Listing 4.1: Naive Bayes Implementation

Accuracy / Output



```
(base) (DWD) $ python3 decision-tree.py
Naive bayes accuracy:  0.8583646685836467
(base) (DWD) $
```

Figure 4.1: Naive Bayes Accuracy

4.2 Decision Tree

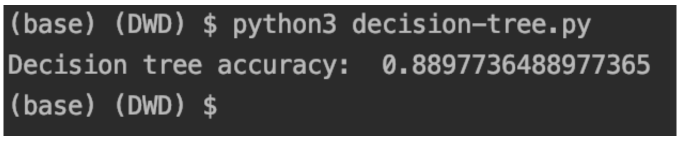
Implementation

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

def decision_tree():
  cal_entropy = DecisionTreeClassifier(criterion="entropy", random_state=1,
    max_depth=3, min_samples_leaf=5)
  cal_entropy.fit(X_train, Y_train)
  Y_pred = cal_entropy.predict(X_test)
  print('Decision tree accuracy: ', metrics.accuracy_score(Y_test, Y_pred))
```

Listing 4.2: Decision Tree Implementation

Accuracy / Output



```
(base) (DWD) $ python3 decision-tree.py
Decision tree accuracy:  0.8897736488977365
(base) (DWD) $
```

Figure 4.2: Decision Tree Accuracy

4.3 Random Forest

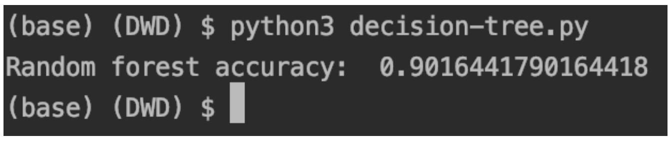
Implementation

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

def random_forest():
  # Create the model with 100 trees
  random_forest_model = RandomForestClassifier(n_estimators=100, bootstrap=
   True, max_features='sqrt')
  # Fit on training data
  random_forest_model.fit(X_train, Y_train)
  Y_pred = random_forest_model.predict(X_test)
  print("Random forest accuracy: ",metrics.accuracy_score(Y_test,Y_pred))
```

Listing 4.3: Random Forest Implementation

Accuracy / Output



```
(base) (DWD) $ python3 decision-tree.py
Random forest accuracy:  0.9016441790164418
(base) (DWD) $
```

Figure 4.3: Random Forest Accuracy

4.4 K-Means

Implementation

Error Diagram

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

def kmeans():
  # Elbow method for number of clusters
  Error = []
  for i in range(1, 11):
    kmeans = KMeans(n_clusters=i).fit(X_test)
    kmeans.fit(X_test)
    Error.append(kmeans.inertia_)
  plt.plot(range(1, 11), Error)
  plt.title('Elbow method')
  plt.xlabel('No of clusters')
  plt.ylabel('Error')
  plt.show()
```

Listing 4.4: K-Means Error Diagram Plotting

Diagram

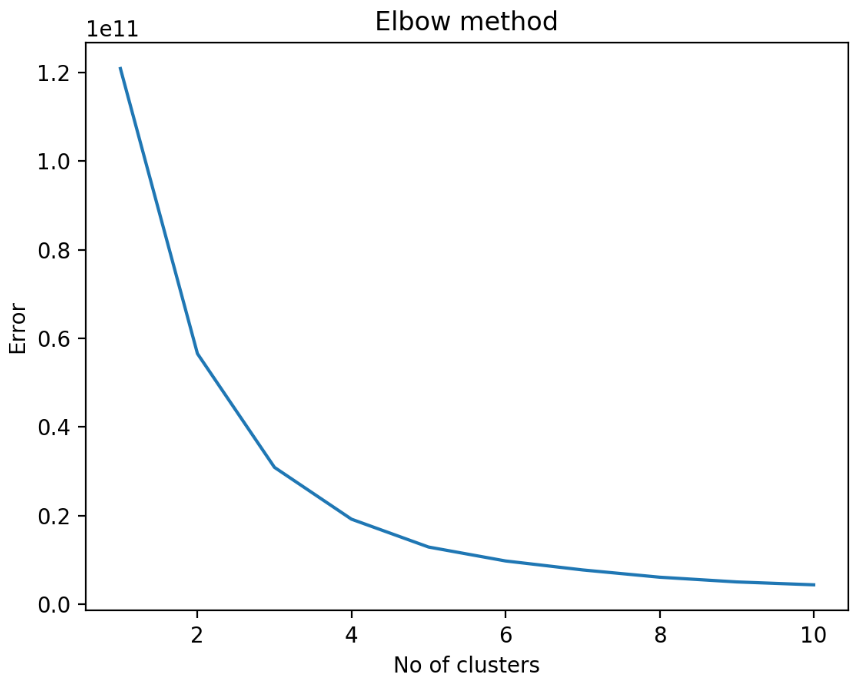


Figure 4.4: Error Diagram

Clustering Implementation

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

def kmeans():
  # kmeans implementation with 3 clusters
  kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
  y_kmeans = kmeans.fit_predict(X_test)
  # visualising kmeans
  plt.scatter(X_test.iloc[:,0], X_test.iloc[:,1],c=y_kmeans,cmap='rainbow')
  plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
     s=300, c='yellow', label='Centroids')
  plt.title("Clusters")
  plt.xlabel("X-label")
  plt.ylabel("y-label")
  plt.legend()
  plt.show()
```

Listing 4.5: K-Means Implementation

Visualising K-Means

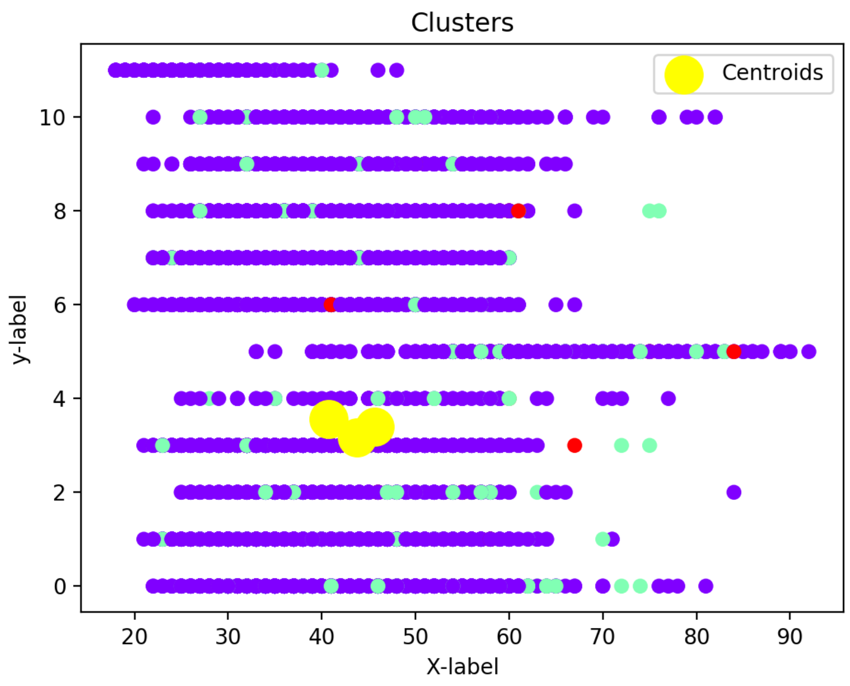
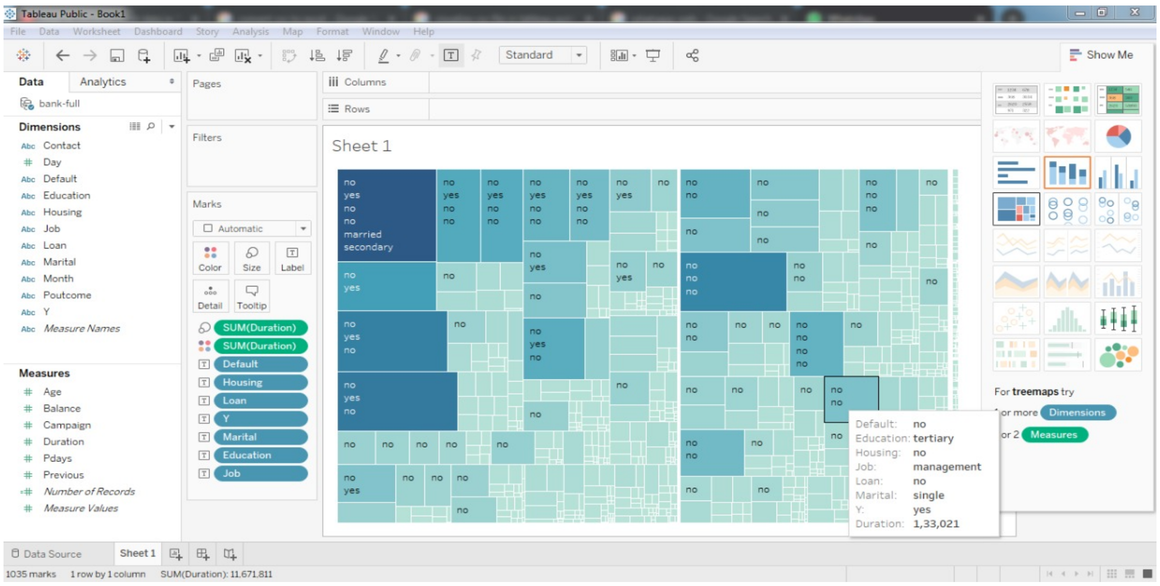


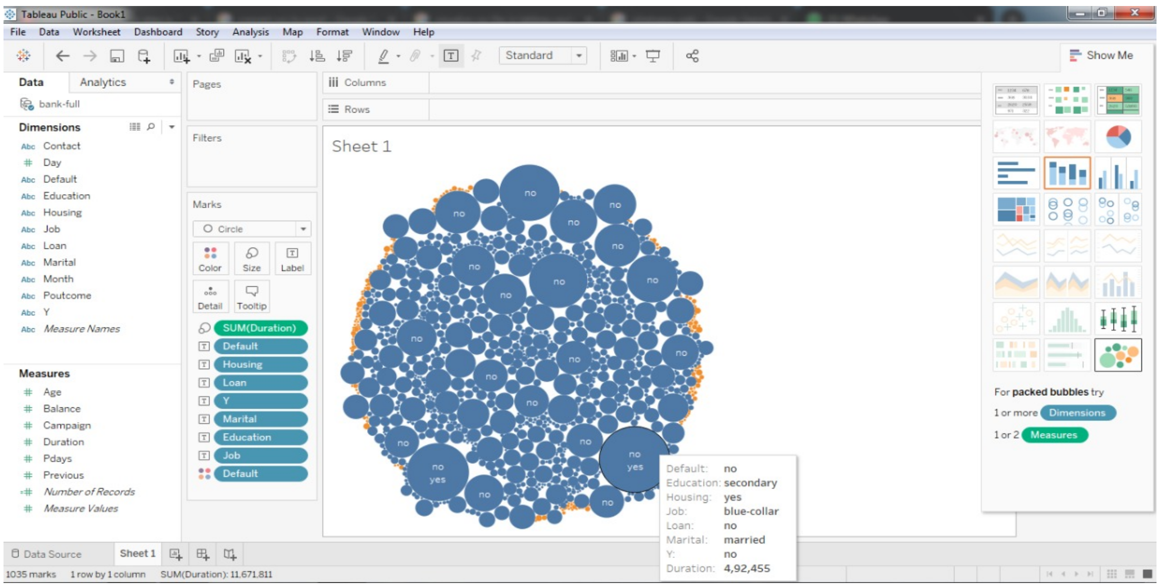
Figure 4.5: K-Means Visualisation

4.5 Tableau



Default: no
Education: tertiary
Housing: no
Job: management
Loan: no
Marital: single
Y: yes
Duration: 1,33,021

Figure 4.6: Tableau Visualization 1



Default: no
Education: secondary
Housing: yes
Job: blue-collar
Loan: no
Marital: married
Y: no
Duration: 4,92,455

Figure 4.7: Tableau Visualization 2

Chapter 5

Conclusion

The objective of the project has been achieved. The model is predicting whether a person would subscribe to a term deposit or not. Different methods for data pre-processing were studied and implemented and with a combination of research and trial and error, the most optimum methods were used.

The programming language and technologies used in the making of this project have been studied and used. Different algorithms were used to predict the result, each giving different accuracy. Each algorithm used has its merits and demerits. In some areas an algorithm was lagging while in other areas the algorithm outperformed the others. On comparing the accuracy and implementation condition for the different algorithms, use cases for the different algorithms was realised.

The prediction algorithms (Decision tree, Naive Bayes and Random Forest) implemented were better for this particular use case. The implementation of K-Means proved to be unsatisfactory, since the data was too complicated to signify any result from clustering. K-Means was implemented and the data points were classified into three clusters(optimum number found by elbow method) but the visualisation was no good. The remaining three algorithms gave an accuracy ranging from $\approx 86\%$ to $\approx 90\%$.

Working in a team of people from mixed backgrounds was a learning experience for all team members. People from non-IT background were exposed to the field of data science and where overwhelmed by it. This was a great experience for the members from IT background as well. This was a truly hands-on experience for all members of the group. This project enabled everyone to learn from one another and to learn something new altogether.