



Wholly owned by UTAR Education Foundation
(Co. No. 578227-M)
DU012(A)

Univeristi Tunku Abdul Rahman

Faculty of Information and Communication Technology (FICT)

Bachelor of Computer Science (Hons)

UCCD2063 Artificial Intelligence Techniques

Practical Assignment

Adult Data Set: <https://archive.ics.uci.edu/ml/datasets/Adult>

	Student 1	Student 2	Student 3
Student Name	Tan Jing Jie	Lew Wei Li	Pamela Yong Zi Wei
Student ID	1804560	1803518	1803580
Contribution	100%	100%	100%

Table of Contents

Chapter 1.	Introduction.....	1
1.1	Motivation.....	1
1.2	Objectives.....	1
Chapter 2.	Method and Experiments	2
2.1	Dataset.....	2
2.1.1	Data Collection	2
2.1.2	Data Exploration	2
2.1.3	Data Visualization.....	5
2.2	Overall framework	13
2.3	Implementation Details	14
2.3.1	Data Pre-processing	14
2.3.2	Model Training and Validation.....	16
2.3.3	Tuning and Testing	24
Chapter 3.	Result and discussion.....	26
3.1	Experimental Result	26
3.1.1	Data Pre-Processing	26
3.1.2	The final test set result (after fine tune)	28
3.1.3	Best model in all data set (before and after fine tune)	29
3.2	Performance analysis.....	30
3.2.1	Model.....	31
3.2.2	Fine Tune Analysis	38
3.2.3	Outlier analysis	39
3.3	Model Comparison.....	40
Chapter 4.	Conclusion	42

Chapter 1. INTRODUCTION

1.1 MOTIVATION

Over the past two decades, human dependence on data has inevitably increased with the advanced technology such as data mining and machine learning. With these advanced technologies, humans are able to predict the future events by exploring the hidden patterns of data. Nowadays, data analysis on income statistics plays an important role in business world. With a good grasp of the data, organizations would be able to gain an insight of a full and reliable picture of the general population income.

If a project team were given a task to support the marketing of an investment project targeted at potential customers with at least medium earning abilities. Assuming that this marketing strategy has achieved a high acceptance rate among people who earn more than 50k per year despite of the fact that it requires a high degree of investment per offer. Hence, it is necessary to create a model to identify the potential customers accurately.

1.2 OBJECTIVES

The *adult* dataset was accessed from the University of California Irvine (UCI) Machine Learning Repository extracted by Barry Becker from the 1994 Census database. The objective of this project is to create a supervised machine learning model to solve this classification task in order to predict the people whose annual income exceeds 50k. This project will be contributing in business world for identifying the target market.

Chapter 2. METHOD AND EXPERIMENTS

2.1 DATASET

2.1.1 DATA COLLECTION

The *adult* dataset accessed were initially split into 2 files named as “adult.csv” and “adult_test.csv” respectively. Before data collection, we first imported all the libraries necessary. After that, we declared the column names for all the attributes as there is no header inside the files. We then imported the datasets into pandas DataFrames using the column names defined and separator ‘ *, *’ to trim all the whitespaces before and after the sample data. Moreover, we replaced the ‘?’ values inside the datasets so that number of null values can be examined later. After collecting the data from “adult_test.csv”, we removed the extra ‘.’ behind the income data. Next, the two DataFrames were concatenated and the index was reset to avoid additional columns.

2.1.2 DATA EXPLORATION

By looking at the first 5 samples in the dataset, there are 15 features or columns in total in this dataset. Since our goal is to predict people’s salary which makes *income* feature our target feature. The labels in the income attribute are either “>50K” or “<=50K”. Each sample data collected consists of the demographic information about an individual including *age*, *workclass*, *fnlwgt*, *education*, *education_num*, *marital_status*, *occupation*, *relationship*, *race*, *sex*, *capital_gain*, *capital_loss*, *hours_per_week*, *native_country* and *income*.

Before gaining insights of which features are necessary for our prediction, we first differentiate the datatypes of the attributes. By looking at the first 5 samples in the dataset and a quick description of all attributes, we found that there are 6 numerical and 9 categorical attributes in this dataset:

Numerical attributes:

- i. *age*: represents individual’s age.
- ii. *fnlwgt*: final weight, the amount of people that this sample represents.
- iii. *education_num*: represents the number of years of education in total, which is a continuous representation of education.
- iv. *capital_gain*: represents individual’s capital gains.
- v. *capital_loss*: represents individual’s capital loss.

- vi. *hours_per_week*: represents individual's working hours per week.

Categorical attributes:

- i. *workclass*: represents the employment status.
 - Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- ii. *education*: represents individual's highest level of education.
 - Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- iii. *marital_status*: represents individual's marital status. Married-civ-spouse means civilian spouse whereas Married-AF-spouse means spouse in the Armed Forces.
 - Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- iv. *occupation*: represents an occupation type.
 - Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op- Inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- v. *relationship*: represents individual's role in the family.
 - Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- vi. *race*: represents a person's race
 - White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- vii. *sex*: represents individual's biological sex.
 - Male, Female
- viii. *native_country*: represents country of origin.
 - United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad&Tobago, Peru, Hong, Holand-Netherlands.
- ix. *income*: label of whether an individual's income exceeds 50k or not.
 - <=50K, >50K

Next, we were able to tell that the datasets consist of 48,842 sample data and 15 columns in total by looking at a quick description of all attributes in the datasets. There are 3 attributes workclass, occupation and native country with missing values which require data cleaning afterwards. By getting the statistics of all numerical attributes in the pandas DataFrame, we were able to see the range of scales of the features are quite different. This highlights the need for standardization to be performed on the data. Furthermore, we were able to detect the presence of outliers in the datasets such as age, capital gain and hours per week. However, we need to visualize the data to gain a better and deeper understanding of the data before being able to pinpoint the outliers for real. Last but not least, we also managed to detect repeated data in the datasets.

```
adult.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   48842 non-null  int64
1   workclass             46043 non-null  object
2   fnlwgt               48842 non-null  int64
3   education             48842 non-null  object
4   education_num         48842 non-null  int64
5   marital_status        48842 non-null  object
6   occupation            46033 non-null  object
7   relationship          48842 non-null  object
8   race                 48842 non-null  object
9   sex                  48842 non-null  object
10  capital_gain          48842 non-null  int64
11  capital_loss          48842 non-null  int64
12  hours_per_week        48842 non-null  int64
13  native_country        47985 non-null  object
14  income                48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Quick description of datasets.

```
adult.isnull().sum()

age                0
workclass          2799
fnlwgt             0
education           0
education_num       0
marital_status     0
occupation         2809
relationship        0
race               0
sex                0
capital_gain        0
capital_loss        0
hours_per_week      0
native_country     857
income             0
dtype: int64
```

Missing values.

2.1.3 DATA VISUALIZATION

Numerical Attributes

After data exploration, we proceed to do data visualization of numerical attributes. Based on Diagram 2.1.3.1, it is noticed that the histograms of the attributes *age*, *capital_gain*, *capital_loss* and *fnlwgt* are tail-heavy which indicates that there is a skewness of data such that all the four attributes mentioned are rightly skewed. Besides that, all the scales are different and thus we need to do standardization on the numerical attributes later.

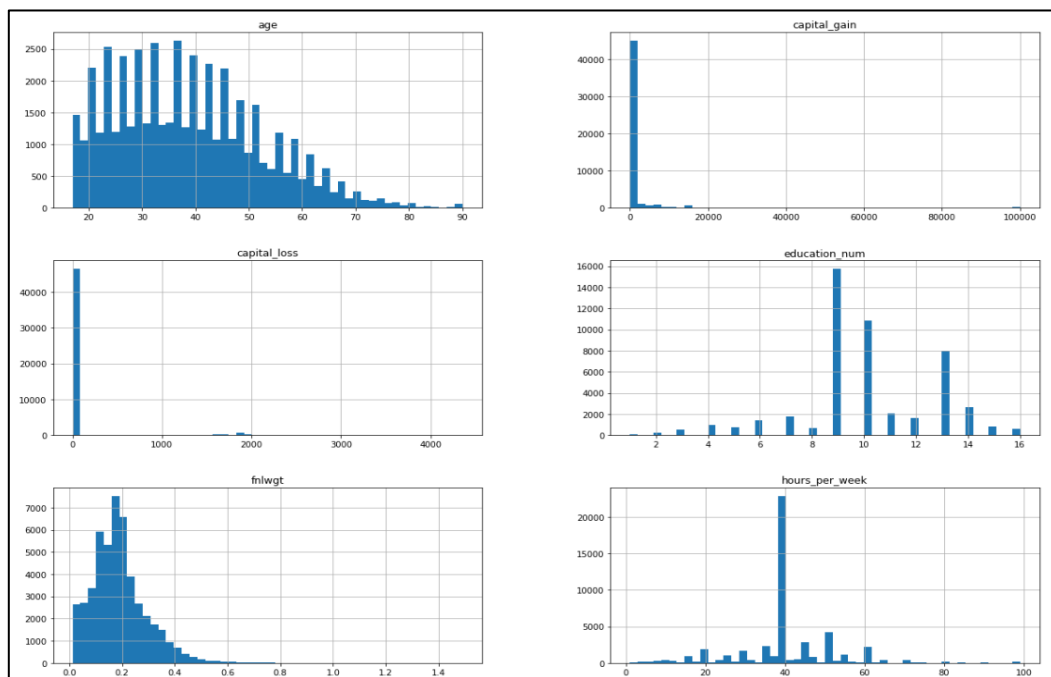


Diagram 2.1.3.1: Histograms of numerical attributes.

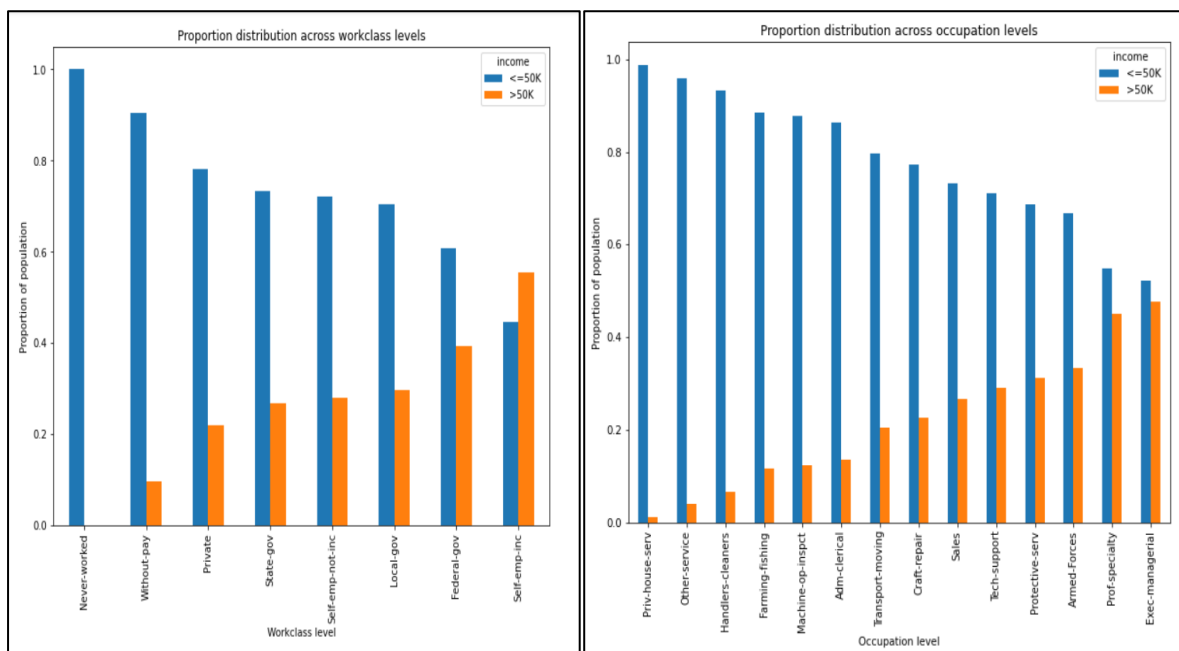
With the use of heatmap, it is noticed that there is no particular strong correlation among the numerical attributes which means that there are no strong positive or negative correlations. Based on Diagram 2.1.3.2, *education_num* and *hours_per_week* show positive correlation (0.14). Besides that, there is also a positive correlation (0.13) between *education_num* and *capital_gain*. However, both correlations are not even close to +1 and thus we cannot conclude the features importance just from here.



Diagram 2.1.3.2: Heatmap showing Correlations between Numerical Data (the brighter the colour, the stronger the correlation).

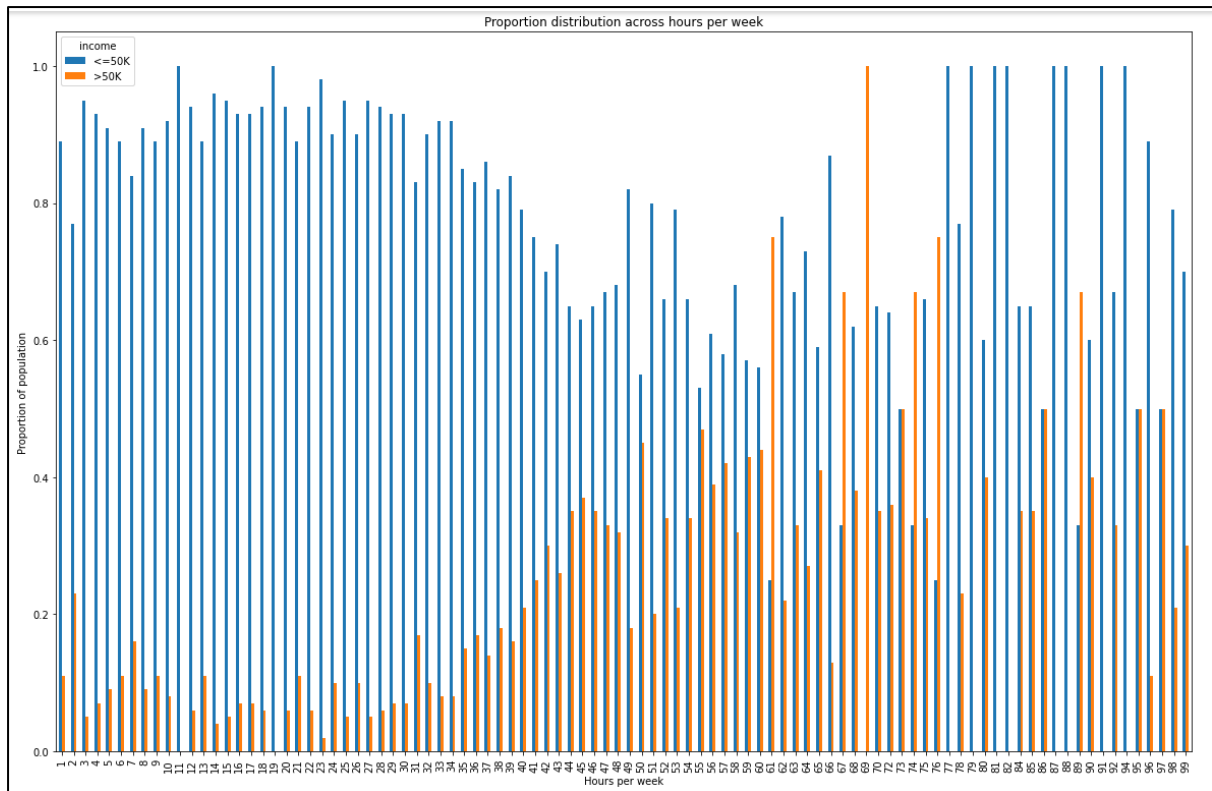
Categorical Attributes

Next, bar charts were also generated as shown below to provide an overview of how the income distributes among the categorical attributes in the dataset. When analyzing the relationship between workclass and income, people who never worked did not earn more than 50k. Only *Self-emp-inc* workclass is having more people earning more than 50k. In every occupation, people who earn less than 50k is more than those who earn more than 50k. Besides that, there is a trend on the relationship between hours worked per week and income, the higher the hours worked per week, the higher the proportion of population making more than 50k a year. However, after 69 hours worked per week, the population earning more than 50k a year starts to drop. Moreover, we can also infer that higher education may result in higher income as people who graduate from Masters, Prof-school or Doctorate are gaining more than those who are having lower education level. When analyzing the relationship between gender and income, males earn more than females in general. Among all the races, White and Asian-Pac-Islander are more likely to earn more than 50k compared to all the other races. Lastly, we can infer that there is no dependency of native country on target attribute income since no useful information could be obtained.

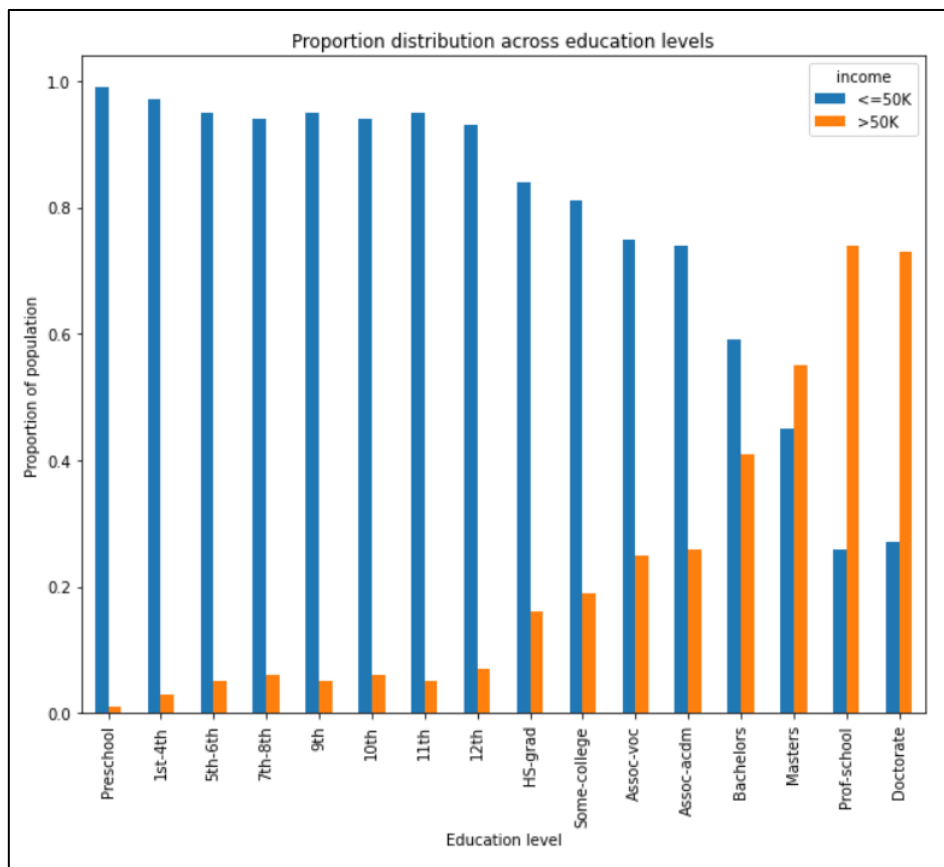


Workclass vs Income

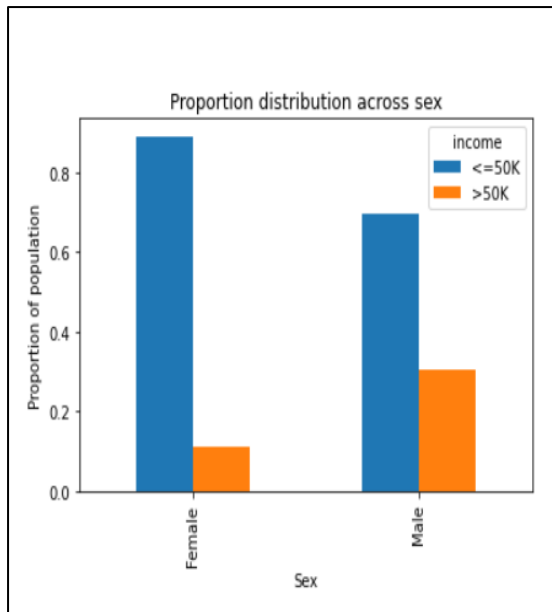
Occupation vs Income



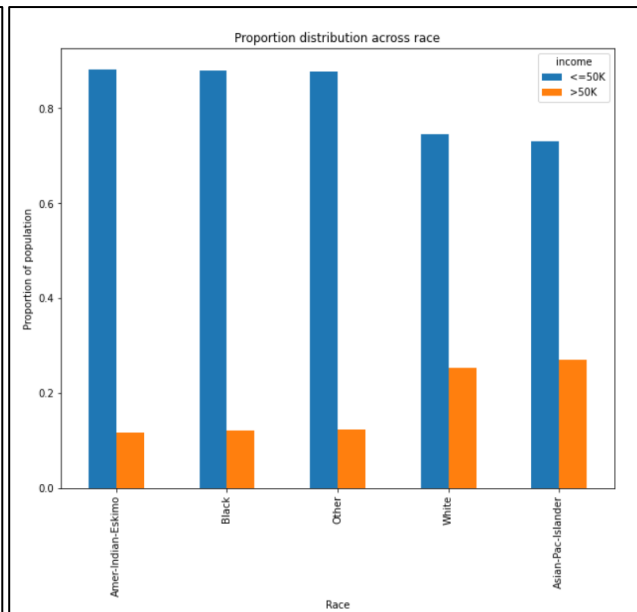
Hours per Week vs Income



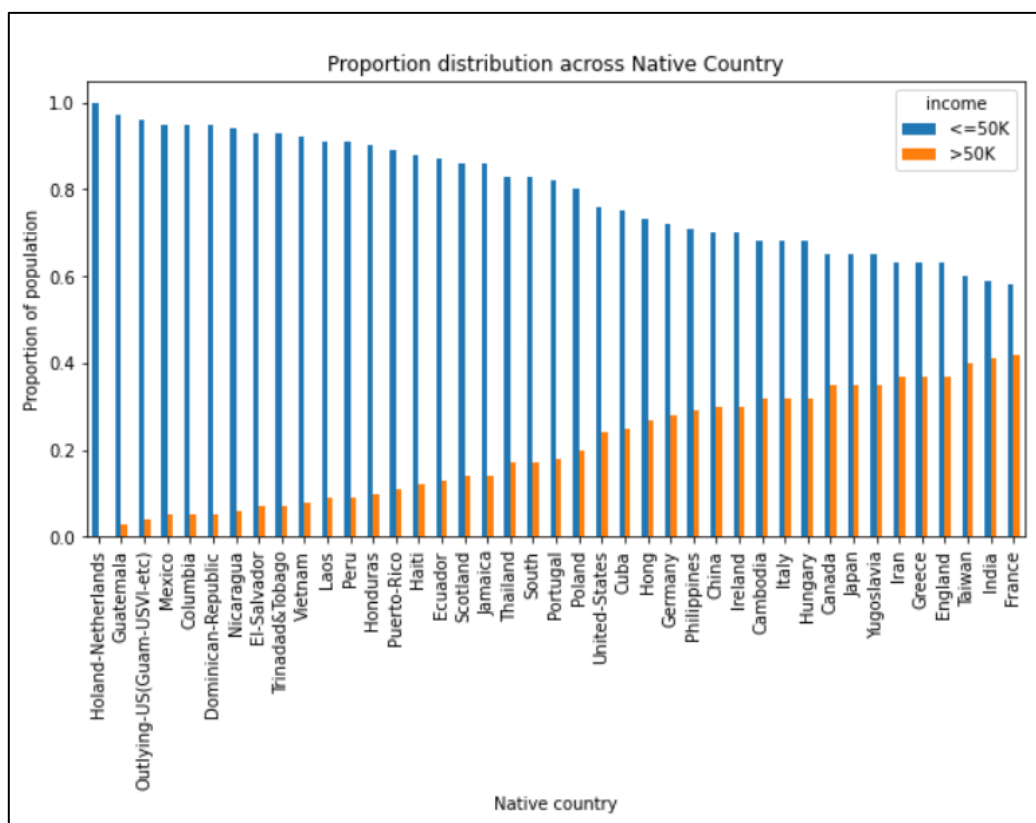
Education vs Income



Sex vs Income



Race vs Income



Native Country vs Income

Outliers visualization

When analyzing the relationship between age and capital gain, we plot two graphs in order to see the full picture and analyze deeper at the same time. Firstly, capital gain of 99999 (the exact value was obtained from data exploration) is clearly an outlier as being shown in Diagram 2.1.3.3. Moving on to Diagram 2.1.3.4, majority of people between the age of 28 and 64 are earning a maximum capital gain of 15k. After that it decreases until a sudden increase at the age of 90. Minority of people between the age of 28 and 59 are earning more than 25k. After that it decreases until a sudden increase at the age of 90 again. With these being said, it could be seen that the age of 90 does not follow the general trend.

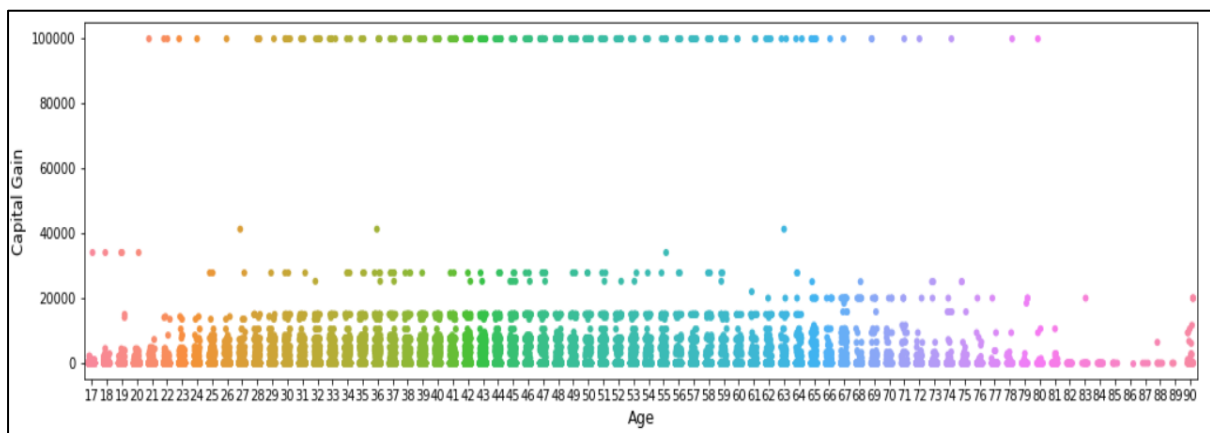


Diagram 2.1.3.3: Capital Gain vs Age (full graph).

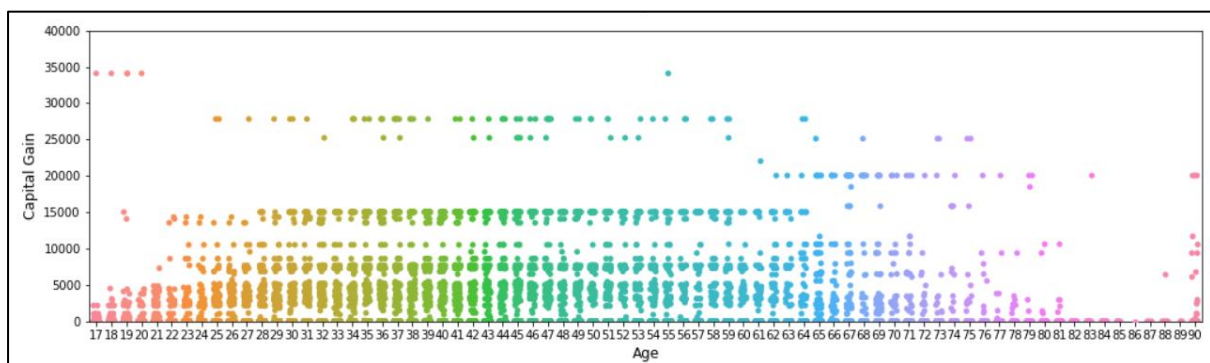


Diagram 2.1.3.4: Capital Gain vs Age (sub graph).

Similarly, capital gain of 99999 is clearly an outlier in Diagram 2.1.3.5 as well. Based on Diagram 2.1.3.6, majority of people work for 40, 50 and 60 hours a week. The capital gain is increasing when hours worked per week increases but started to decrease after 60 hours worked per week. There are a few people working for 99 hours per week but their capital gain is less than those who only work for 40 hours a week. Therefore, we can also safely conclude

that 99 hours worked per week is an outlier. Moreover, we can also infer that hours per week exceeding 99 hours are all outliers as people either have to work for more than 20 hours if they only work on weekdays or they need to work for more than 14 hours every day including weekends, which is not logical.

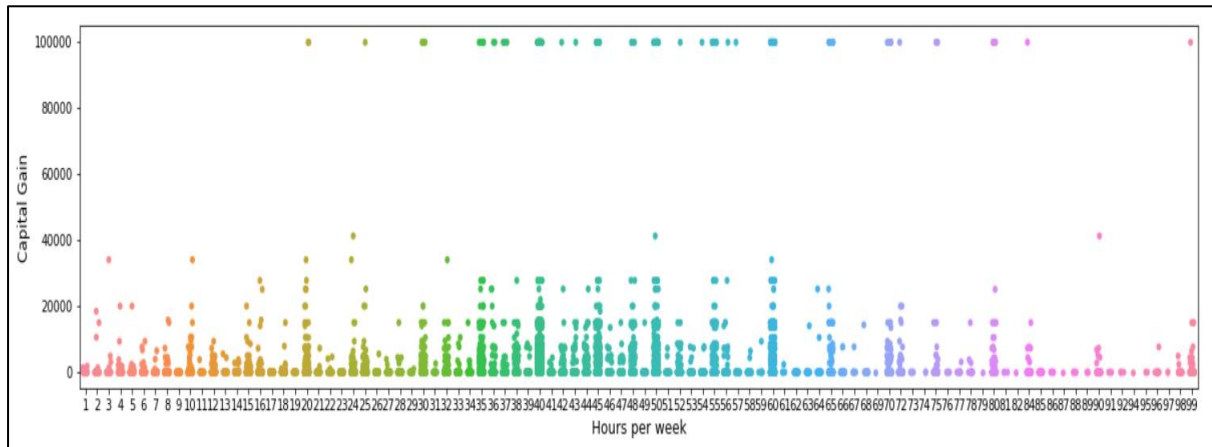


Diagram 2.1.3.5: Hours per week vs Capital Gain (full graph).

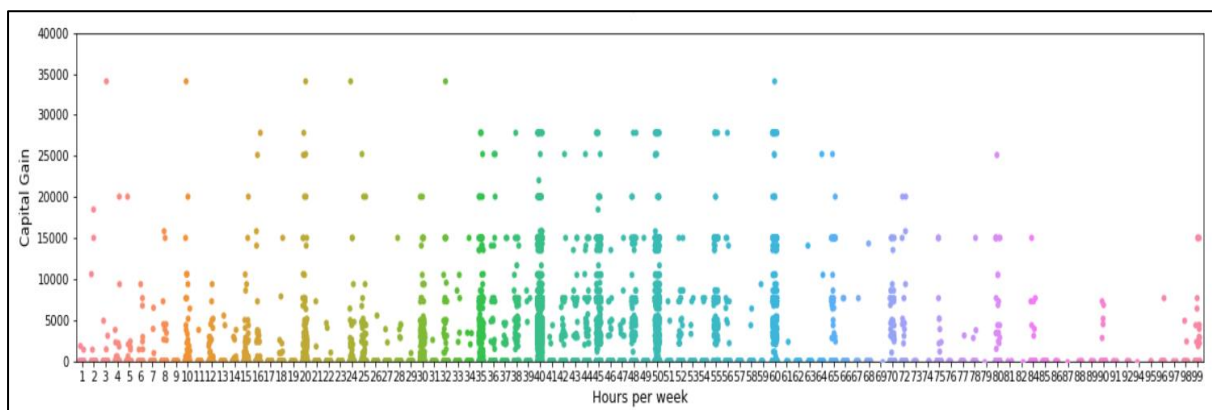


Diagram 2.1.3.6: Hours per week vs Capital Gain (sub graph).

Based on Diagram 2.1.3.7, the age of 90 is clearly an outlier as it does not follow the general pattern at all. When examining the outliers, it is simply not logical to have a person at the age of 90 working for 99 hours a week such that he or she would need to work for 14 hours every day.

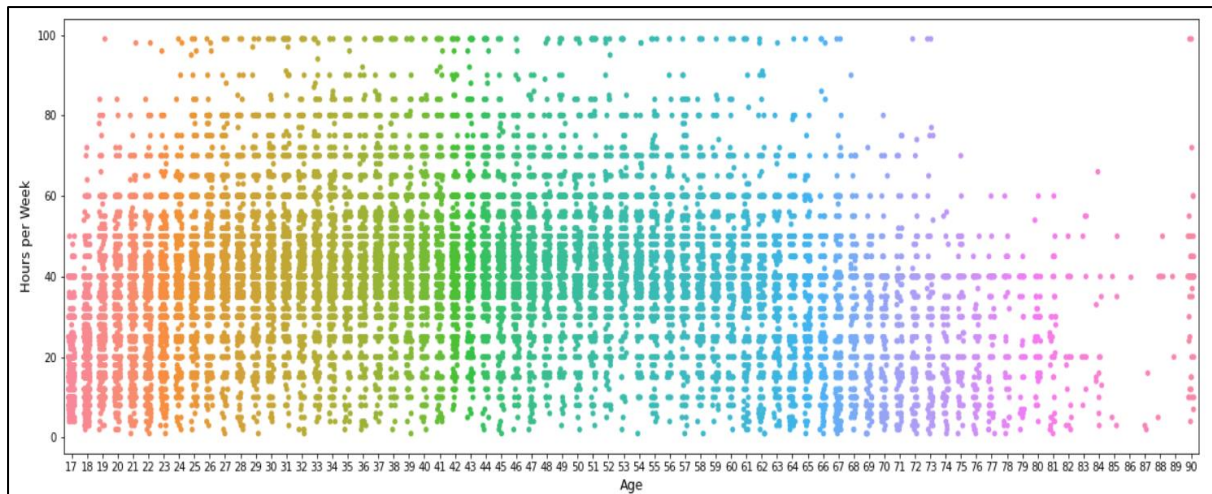
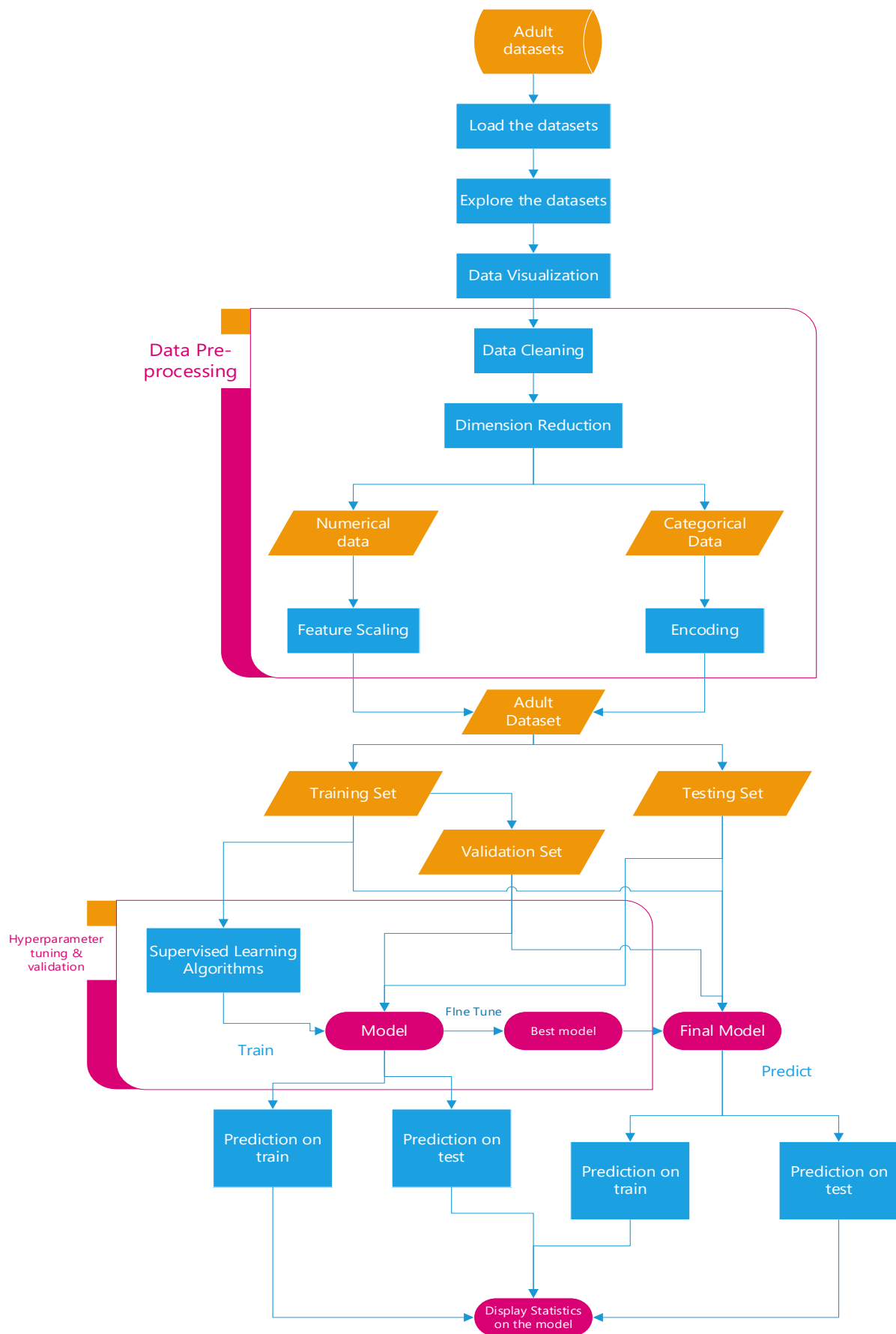


Diagram 2.1.3.7: Age vs Hours per week.

2.2 OVERALL FRAMEWORK



2.3 IMPLEMENTATION DETAILS

2.3.1 DATA PRE-PROCESSING

In this project, we apply the following data pre-processing techniques to perform data cleaning prior to data modelling:

1. Removing repeated values
2. Outliers removal
 - Capital gain of 99999
 - Age more than or equal to 90
 - Hours worked per week more than or equal to 99
3. Handling missing values
 - Working class
 - Occupation
 - Native country
4. Removing useless columns
 - Final weight
 - Education
 - Relationship
5. Data Transformation
 - i. Processing numerical data – feature scaling
 - ii. Processing categorical data – encoding

Since the duplicated data are in a small size which is only 52 records, we can remove the duplicated data directly and keep only the first occurrences. The reason is that, if we did not remove the repeated rows, our algorithm model will be facing an issue – weightage. The duplicated rows can cause the probability of algorithm model to give an unnecessary weightage of an attribute.

We then proceed to remove the outliers in order to enhance the performance of our models later. Any values with capital gain of 99999, the age of more than 90 or hours per week more than 99 were dropped. As a result, 432 records were removed which may affect the performance significantly and it will be examined in the later section – [3.0 Results and Discussion](#).

Next, missing values were handled after removing the repeated row and outliers. We filled the missing values in each of the three columns (*workclass*, *occupation*, *native_country*) by predicting their values. For each of the three columns, we used all the other attributes (including *income*) as input features and treat the column as the targeted variable. Making it a multi-class classification task. We used two different classification algorithms, which is Decision Trees and Random Forest to predict the class when the value is missing. Then by taking a majority vote amongst the two classifiers (use the value to fill the missing column if both classifiers have the same value, else the majority class (mode) of that column will be chosen for the missing value column). We chose not to simply drop the missing values because there is a total of 6454 missing values in the datasets which may cause significant effect in model performance.

Going into the details of handling missing values, we first extracted rows from the original DataFrame (*adult*) with no missing targeted variable into a new DataFrame (*sample_data*) and rows with missing targeted variable into another DataFrame (*missing_data*) with `copy()` function so it won't affect the original DataFrame. Then, we also get the targeted variable column without missing value into a new variable (*sample_label*). Then for both DataFrame we just dropped the targeted variable column as we were doing a classification task. Then we identified the categorical column to separate the data into numerical feature and categorical feature, and perform one hot encoding on the categorical feature and then joined back the numerical and categorical features back to one DataFrame, this action was done to both the DataFrame. After the data is ready to be fit, we generated a Decision Tree and Random Forest classifier object and fit it with (*sample_data_tr*) which is DataFrame without missing values and with the targeted variable(*sample_label*). Then we used the classifier object to predict on target variable on the row with missing column (*missing_data_tr*). Next, we detected what is the majority vote (mode) in the DataFrame, then we put the predicted result into a DataFrame (*pred_df*), then by majority vote, we chose the best value for particular column. Lastly, we replaced the column in the *adult* DataFrame with '?' as missing value with the chosen predicted value(*overall_pred*).

Before splitting the *adult* DataFrame into input and output matrix, we first dropped the useless columns which were *fnlwgt*, *education* and *relationship* respectively. As a result, 12 columns were left in the DataFrame. After that, we split *adult* into an input matrix *X* (all the 11 attributes except *income*) and an output matrix *y* (*income*) and then the input matrix was further split into numerical and categorical sets for data transformation. Since some models can only

handle numerical data, we then converted the categorical data into numerical data using pandas *get_dummies* which can perform similar task like one-hot encoding, both are used to encode nominal data. In this case, LabelBinarizer cannot be used as it is not suitable for multiple outputs. As mentioned earlier, the numerical data are having different scales, so we performed standardization on numerical data for feature scaling. During standardization, the mean and standard deviation were computed for each attribute and stored. All the numerical attributes were then standardized using the computed mean and standard deviation values.

Lastly, we combined the numerical attributes and categorical attributes back together again after performing data cleaning on them. The dataset was then finalized after the train test split (80% train set, 20% test set).

2.3.2 MODEL TRAINING AND VALIDATION

The Scikit learn package in Python offers a wide variety of classifier models including SGD, SVM, Neural Network, Ensemble classifiers and etc. Our approach is to try various methods on a trial basis to determine which model performs the best.

Strategy:

- Try out 8 different models from various categories of machine learning algorithms.
 - i. AdaBoost
 - ii. Artificial Neural Network (ANN)
 - iii. Decision Tree Algorithm
 - iv. K Nearest Neighbors (KNN)
 - v. Logistic Regression
 - vi. Random Forest Regressor
 - vii. Stochastic Gradient Descent (SGD)
 - viii. Support Vector Machine (SVM)
- Do not spend too much time tweaking the hyperparameters.

Firstly, we built the models and used the trained models to make predictions using the training samples where our baseline accuracy and performance could be obtained here. According to our strategy stated above, we used default hyperparameters to save our time and also gain a better understanding about the nature of the algorithms in this dataset but we will still find the best hyperparameters for each model in the later section – [2.3.3 Tuning and Testing](#). After that, we evaluated each model using cross-validation to avoid overfitting issues. During

the cross-validation, the original training data were split into 3 folds. One of the three was treated as a validation set whereas the other two as training sets. The *cross_val_score* function was set to return the accuracy score. Besides validation, multiple metrics for performance measure were also computed including accuracy, precision, recall, F1 score, sensitivity, specificity, error rate, mean cross validation score and Area Under Curve (AUC) score.

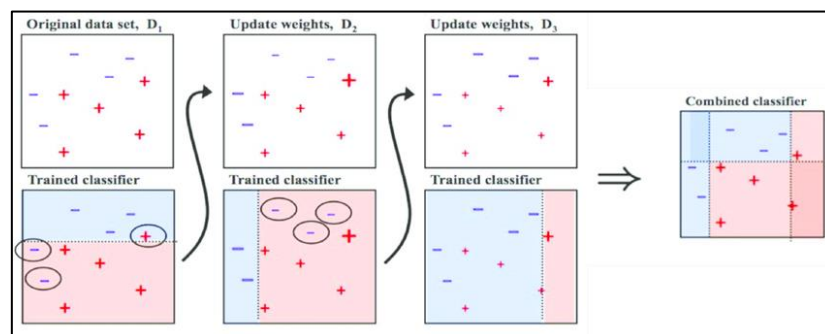
Justifications for Selection and Implementation

A. AdaBoost

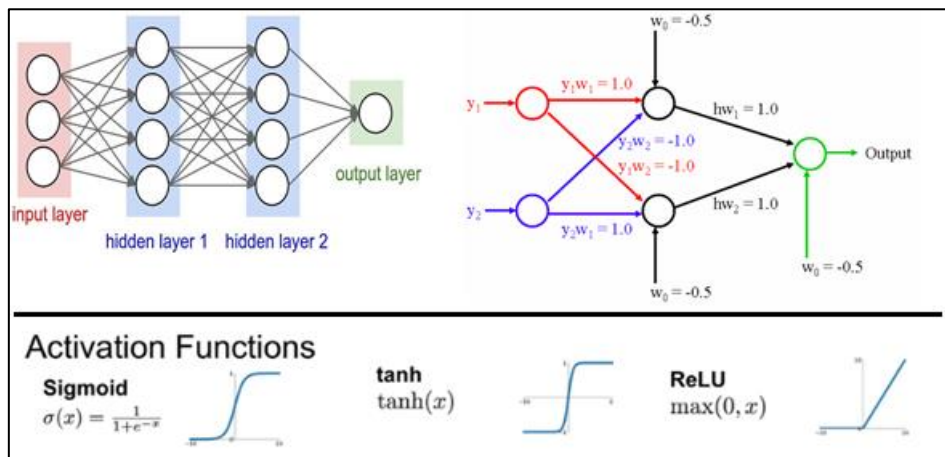
AdaBoost is an ensemble learning method which is known for increasing the efficiency of binary classifiers. AdaBoost uses iterations to learn from the mistakes of weak classifiers and transform them into strong ones. In other words, AdaBoost can be used to improve the accuracy of the weak classifiers. One of the advantages of AdaBoost is that it does not require that much hyperparameters tweaking. Another advantage is that AdaBoost is less prone to overfitting problems when compared to other algorithms. The drawback of AdaBoost is that it is very sensitive to noise data and outliers. However, we already eliminated outliers during data pre-processing so this shall not be a concern.

At begin, AdaBoost will randomly select a training subset. AdaBoost machine learning model iteratively by select the training set based on the accurate prediction of its last training. The higher weight will be classified to wrong classified observations so that these observations will get more more classification in next iteration in term of probability. Moreover, AdaBoost also will assign some weight to trained classifier in each iteration according to the level of accuracy of correspond classifier. The higher the accurate classifier, the higher the weight. All this process will iterate again and again until the complete training data set which fit and without any error. On the other hand, if it reached to the specified maximum number of estimators, it will also be terminated.

For explainable, this picture will further enhance the understanding:



B. Artificial Neural Network (ANN)



ANN is a well-known algorithm. It is a computing system which look same as human brain analyse and processes information. Hidden layer means that there are many repeated “outputs” in between and use as input. Repeat again the process until a final output is obtained. After fine tune, we use identity instead of Rectified Linear Unit (ReLU) as a piecewise linear function.

Why we non-linear activation function such as sigmoid and tanh function is not considered? First, we need to discuss why ReLU is better, this is because the limitation of hardware capability. The layer deep in large network fail to receive useful gradient information. The error is propagated through the network and dramatically propagated this error to following layer. ReLU is a linear function allowing complex relationship to be learned. Moreover, ReLU is simple and fast comparing to non-linear function.

Now we can further explain the use of Identity activation function. Identity activation function is a no-op activation and it is useful to prevent linear bottleneck (The shortage of ReLU activation function).

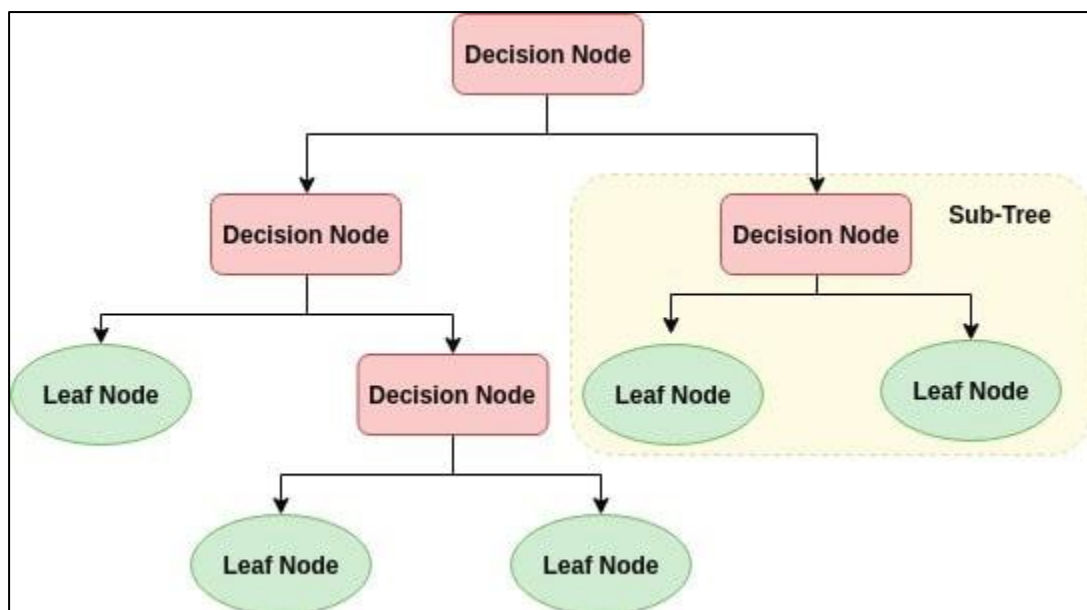
We choose Ann as it has the ability to learn and model non-linear and complex relationships, which is better as many of the relationships between inputs and outputs are nonlinear as well as complex. Next, ANN can generalize well on unseen data and thus making the model generalize and predict on unseen data Lastly, Ann doesn’t not impose and restrictions on the input variables, many studies have shown that ANN can deal with data with high volatility and non-constant variance.

C. Decision Tree Algorithm

Decision Tree is a supervised learning algorithm made up of the if-then-else rule. Less data preparation is required such as normalization of data, feature scaling, handling missing values and etc. during pre-processing when compared to the other algorithms. However, the structure of Decision Tree would easily become unstable when there is only a small change in the data. Nevertheless, we still chose Decision Tree to compare its performance with Random Forest.

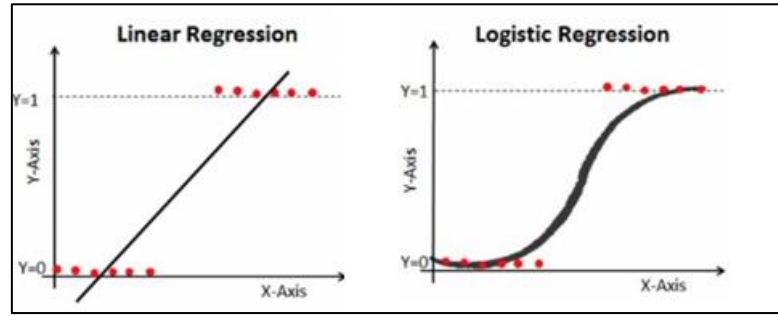
We choose decision tree because compared to other algorithms decision tree requires less effort for data preparation during pre-processing, as a decision tree does not need require normalization of data and scaling of data. Decision are capable of handling both continuous and categorical variables.

In short, the decision tree work as follow:



The decision tree will use the training set to build it own tree, and use the tree for prediction. The benefit is, it is very easy to understand and implement, but it led to a shortage – the tree is very non-robust. This is because a small change in the training data set also will be led to a large change to the build tree and lead to prediction error.

D. Logistic Regression



Logistic regression is almost same as linear regression algorithm. Logistic regression is using the natural logarithm function to find the relationship between the variables and use the test data to find the coefficients. Logistic regression uses the concept of the odds ratios to calculate the probability. For example, the probability of a badminton team to win a match is 0.80. The probability for that team to lose would be $1 - 0.80 = 0.20$. The odds for that team winning would be $0.80/0.20=4$. This can be said as the odds of the team winning are 4 to 1.

We can now explain theory here:

$$Odds = \frac{P(y = 1|x)}{1 - P(y = 1|x)}$$

Thus the logistic equation is created by taking the Odds ratio

$$Logistic(P(x)) = \ln\left(\frac{P(y = 1|x)}{1 - P(y = 1|x)}\right)$$

Then we know that the probability can said to be linear respect to x

$$Logistic(P(x)) = a + bx$$

Slightly doing some mathematical transformation:

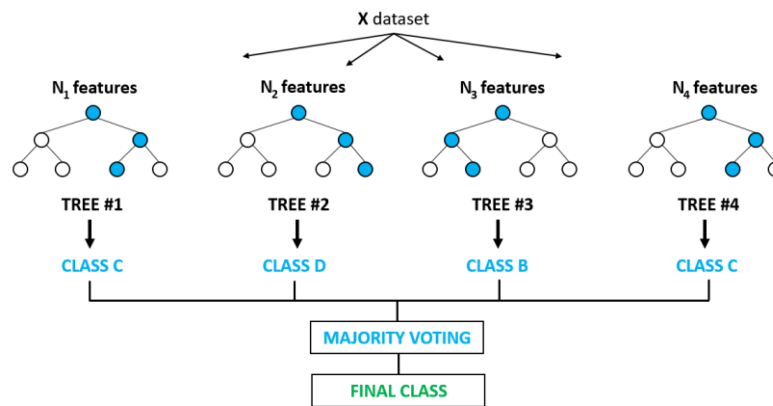
$$\frac{P(y = 1|x)}{1 - P(y = 1|x)} = e^{a+bx}$$

$$P(y = 1|x) = \frac{e^{a+bx}}{1 + e^{a+bx}} = \frac{1}{1 + e^{-a-bx}}$$

Thus, a logistic regression equation is formed for the graph.

We choose logistic regression as it is one of the simplest machine learning algorithms and is easy to implement yet provides great training efficiency. It does not require too many computational resources, input features to be scaled and it is easy to regularize and it performs well when the dataset is linearly separable.

E. Random Forest Classifier



Random forest is a machine learning algorithm made up of multiple decision trees such that different decision trees are not correlated at all. The conclusion is based on the mode of result of many decision trees. This algorithm can be used for both classification and regression problems. We chose random forest because it can judge the importance of the features and their interactions with each other. Moreover, random forest balances the error for unbalanced datasets and do not overfit easily which could be useful in our project since the dataset is not balanced. Deep decision trees may result in overfitting but random forest can prevent overfitting as it creates trees on random subsets.

F. K Nearest Neighbour

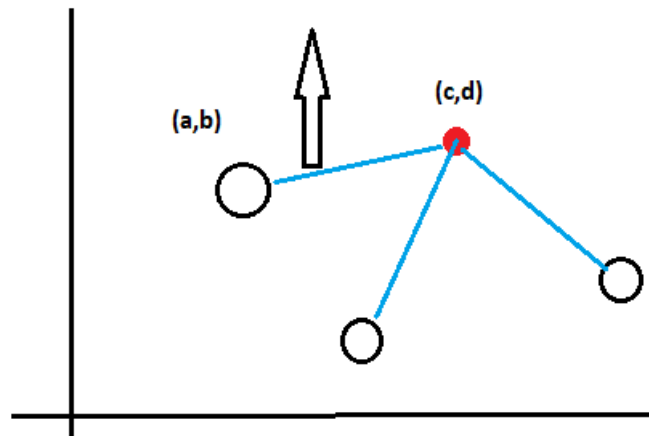


KNN is an algorithm which determine the output by it surrounding. For example as shown in the left picture. The start is the one we want to predict. So first KNN algorithm will look at the nearest neighbour. In this example is red circle. “K” means a number, let say, $k=4$

mean that the 4 nearest neighbours will be selected. And the algorithm will find the mode value. In this case “red circle” is the mode, and it will be classified as red circle in this algorithm.

KNN is use Euclidean distance to classify. The main concept is just a Pythagoras theorem.

$$distance = \sqrt{(c - a)^2 + (d - b)^2}$$



KNN because is very easy to implement. KNN can be said that it “no” learns anything in the training period. After it processes and stores the whole training data set and learns from it only at the time of making real time predictions. Since it does not require no training, new data can be added seamlessly and not affect the accuracy of the algorithm.

G. Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifier under convex loss function. SGD classifier implements a plain stochastic gradient learning routine which support different loss functions and penalties for classification. The default loss function is hinge loss which is equivalent to a linear SVM. The SGD learning, the gradient of the loss is estimated each sample at a time and the model is updated along the way with a learning rate.

$$\begin{aligned} & \text{for } i \text{ in range}(m): \\ & \theta_j = \theta_j - \alpha(\hat{y}^i - y^i)x_j^i \end{aligned}$$

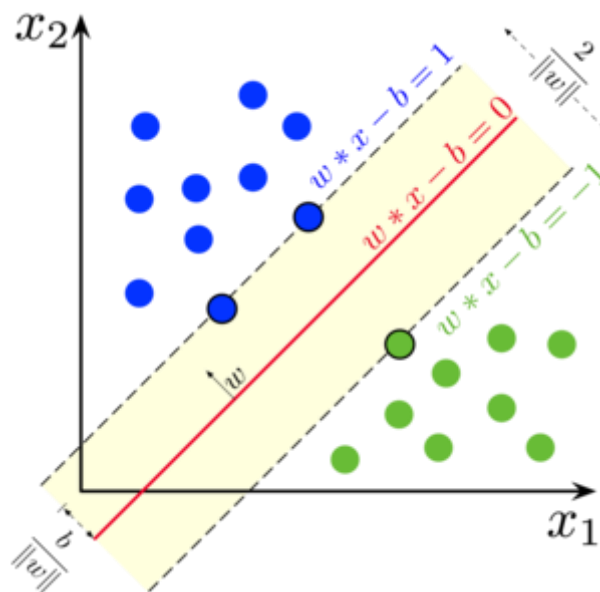
For best result of using the default learning rate, the data should have zero mean and unit variance.

The advantage of using SGD classifier is that it easily scales to problems with more than 10^5 training examples and more than 10^5 features, thus it is suitable for large set of data and it is fast and efficient.

The disadvantage is that SGD is sensitive to feature scaling and it requires a number of hyperparameters such as the regularization parameter and the number of iterations to be tuned to get the best result out of it.

We choose SGD as it can scale very well to a large dataset. And the default parameter is enough for us to get a good result.

H. Support Vector Machine (SVM)



A support vector machine is a supervised machine learning algorithm. In the SVM, we plot each data items as a point in n-dimensional space (where n is the number of features) with the value of each feature being the value of particular coordinate. Then we perform classification by finding the hyper-plane that differentiates the two classes very well.

We choose SVM because it is effective in high dimensional spaces. SVM is versatile different kernel functions can be specified for the decision function to produce a better result. Lastly, SVM is effective in cases where number of features is greater than the number of samples.

2.3.3 TUNING AND TESTING

In this section, we aim to improve the models by analysing the types of errors each of them makes. In our project, we used random search to obtain the best accuracy and the best hyperparameters of each model by testing random combinations of the hyperparameters. Random search is preferred over grid search because grid search is extremely computationally expensive. In reality, we hardly know if we found our best hyperparameters setting unless we have an infinite amount of time to run. In contrast, random search is more efficient when compared to grid search. Although grid search can help us find the optimal value of hyperparameters, random search can help us find close to optimal values in fewer iterations. Therefore, given limited time, our strategy is to use random search. Next, the best hyperparameters obtained from the random search were used to tune the models in order to achieve the optimal performance.

The difference is that, grid search will test all the possibilities while random search only uses “random” method to find out the best hyperparameters. Let see the following picture:

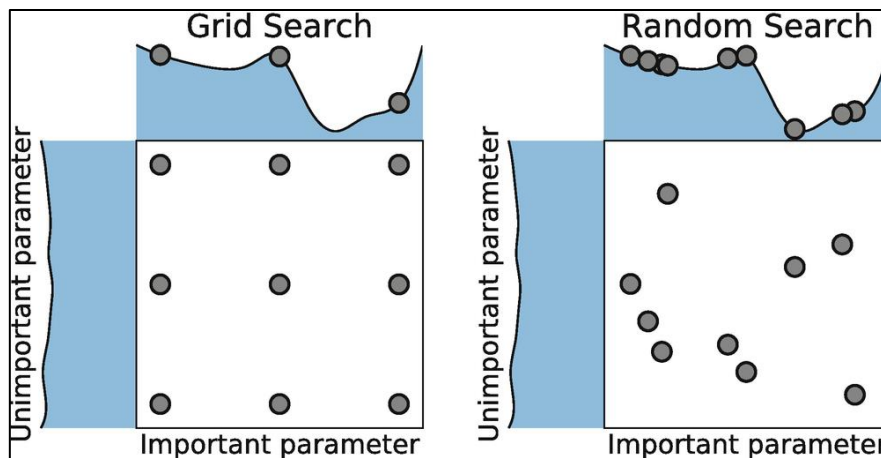


Diagram 2.3.3.1 The difference of grid search and random search visualisation.

For grid search, it is “brute-forcing” to test each combination, but for a random search, it is testing with a random method. The difference is that if we did not specify how many combinations in a random search, its default will be 10 testings’ (n -iteration). In short, if we need to tune a lot of combination, we need to use random search as it is faster and more possible to find out the best parameter in a short time. Mean that in comparison, if grid search using (3×3 hyperparameter). And meanwhile, if random search using ($n \times n$ hyperparameter, where $n > 3$) and with 9 iterations (same with the grid search combination). Random search can be concluded that it has a high chance to get our ideal parameter.

Grid Search	Random Search
Tries all possible combination	Random select a subset of combination within the sample space (and setting the number of iteration)
More computation is needed (more cost)	Less computation is needed (less cost)
Time consuming	Not time consuming
Guaranteed to find the best score from the sample	Not guaranteed to find the best score, but can find the good one in short time)

Thus, with this solution, random search used the same theorem to perform a well-done random search. While when it unable to find the best hyperparameter in searching range, it will be done until the end (n-iteration is set), and choose the best one (may from the reference range), which mean it will look like a grid search (no all the subset as limit by n-iteration).


Chapter 3. RESULT AND DISCUSSION

3.1 EXPERIMENTAL RESULT

3.1.1 DATA PRE-PROCESSING

2.1 Drop repeated values (duplicate rows)

```
[ ] print("Shape before drop:", adult.shape)
    adult = adult.drop_duplicates(keep = 'first')
    print("Shape after drop:", adult.shape)
```

 Shape before drop: (48842, 15)
Shape after drop: (48790, 15)


A total of 52 duplicated records were removed.

2.2 Drop outliers

```
▶ # drop capital gain of 99999
print("Shape before dropping capital gain of 99999:", adult.shape)
adult = adult.drop(adult[adult['capital_gain']==99999].index)
print("Shape after dropping capital gain of 99999:", adult.shape)

# drop age of 90
print("Shape before dropping age of 90:", adult.shape)
adult = adult[adult['age'] < 90]
print("Shape after dropping age of 90:", adult.shape)

# drop 99 hours per week
print("Shape before dropping 99 hours per week:", adult.shape)
adult = adult[adult['hours_per_week'] < 99]
print("Shape after dropping 99 hours per week:", adult.shape)
```

 Shape before dropping capital gain of 99999: (48790, 15)
Shape after dropping capital gain of 99999: (48546, 15)
Shape before dropping age of 90: (48546, 15)
Shape after dropping age of 90: (48492, 15)
Shape before dropping 99 hours per week: (48492, 15)
Shape after dropping 99 hours per week: (48358, 15)

A total of 432 outliers were removed.

Private	35443
Self-emp-not-inc	3794
Local-gov	3126
State-gov	1974
Self-emp-inc	1620
Federal-gov	1429
Never-worked	951
Without-pay	21
?	0

Name: workclass, dtype: int64

Craft-repair	7964
Prof-specialty	6121
Exec-managerial	6055
Adm-clerical	5889
Sales	5528
Other-service	5079
Machine-op-inspct	3066
Transport-moving	2381
Handlers-cleaners	2096
Farming-fishing	1487
Tech-support	1451
Protective-serv	979
Priv-house-serv	247
Armed-Forces	15
?	0

Name: occupation, dtype: int64

Workclass after filling in missing values.

Occupation after filling in missing values.

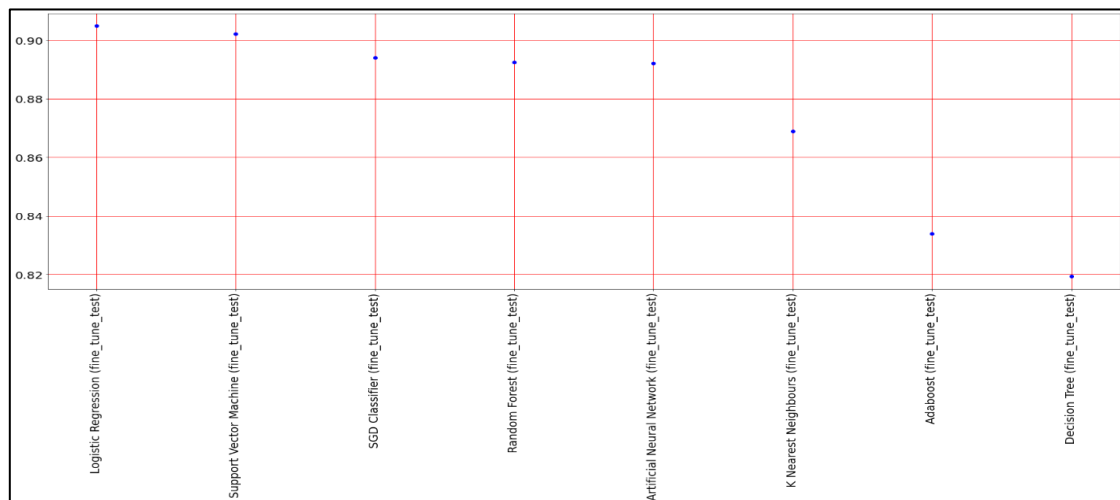
United-States	44200
Mexico	947
Philippines	300
Germany	206
Puerto-Rico	184
Canada	181
El-Salvador	155
India	148
Cuba	137
China	126
England	125
South	117
Jamaica	107
Italy	105
Dominican-Republic	102
Japan	95
Vietnam	88
Poland	87
Guatemala	86
Columbia	84
Haiti	75
Portugal	68
Taiwan	66
Iran	59
Greece	49
Nicaragua	49
Peru	46
Ecuador	44
Ireland	37
France	37
Hong	33
Thailand	30
Cambodia	28
Trinidad&Tobago	27
Yugoslavia	23
Laos	23
Outlying-US(Guam-USVI-etc)	23
Scotland	21
Honduras	20
Hungary	19
Holand-Netherlands	1
?	0

Name: native_country, dtype: int64

Native Country after filling in missing values.

3.1.2 THE FINAL TEST SET RESULT (AFTER FINE TUNE)

Graph



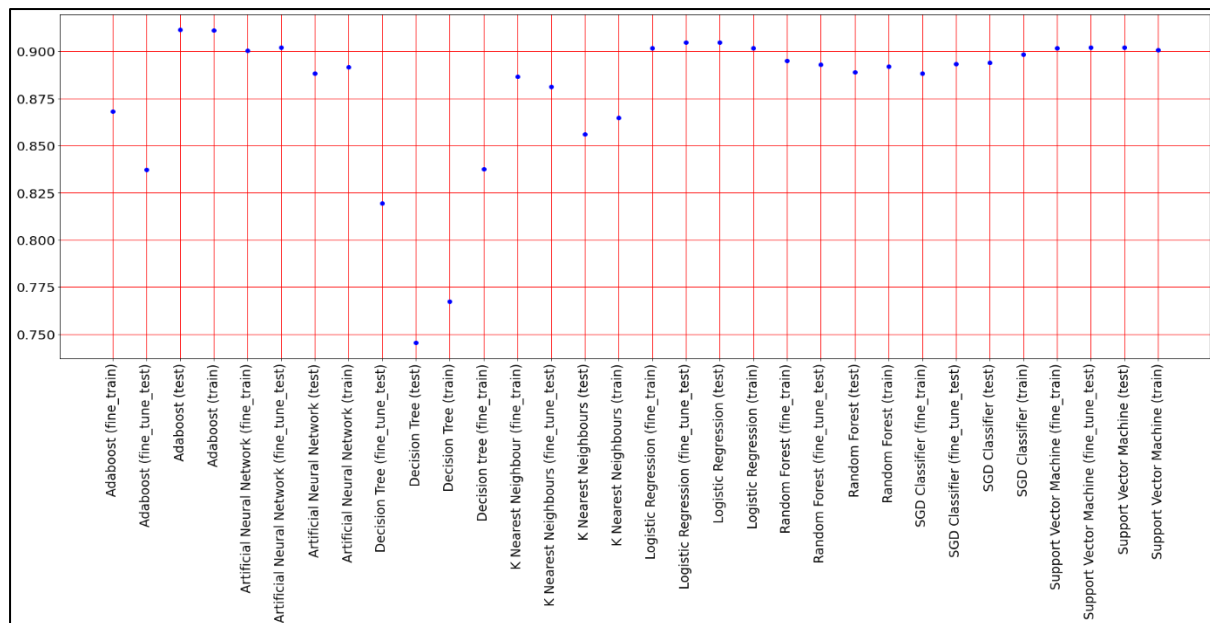
Result (Note “-1” mean there are no training time, it is undergo only a prediction)

	accuracy	precision	recall	F1_score	sensitivity	specificity	error_rate	mean_cross_val_score	auc_score	train time	pred time
Logistic Regression (fine_tune_test)	0.8546	0.7247	0.6049	0.6594	0.6049	0.9303	0.1454	0.8528	0.9047	-1	0.0010
Artificial Neural Network (fine_tune_test)	0.8537	0.7344	0.5813	0.6490	0.5813	0.9363	0.1463	0.8514	0.9022	-1	0.0239
Support Vector Machine (fine_tune_test)	0.8554	0.7688	0.5409	0.6350	0.5409	0.9507	0.1446	0.8530	0.9020	-1	12.7355
SGD Classifier (fine_tune_test)	0.8413	0.8150	0.4111	0.5465	0.4111	0.9717	0.1587	0.8346	0.8933	-1	0.0010
Random Forest (fine_tune_test)	0.8569	0.7159	0.6382	0.6748	0.6382	0.9232	0.1431	0.8471	0.8930	-1	0.5286
K Nearest Neighbours (fine_tune_test)	0.8522	0.6988	0.6404	0.6684	0.6404	0.9163	0.1478	0.8419	0.8812	-1	16.5723
AdaBoost (fine_tune_test)	0.8450	0.6786	0.6342	0.6556	0.6342	0.9089	0.1550	0.8209	0.8374	-1	0.1322
Decision Tree (fine_tune_test)	0.8452	0.6878	0.6129	0.6482	0.6129	0.9157	0.1548	0.8319	0.8193	-1	0.0050

We will focus on the yellow-highlighted part as suits with our objective – High recall, F1 score and AUC score and short prediction time. Explanation: [3.2 Performance Analysis](#).

3.1.3 BEST MODEL IN ALL DATA SET (BEFORE AND AFTER FINE TUNE)

Graph



Overall Result (Note “-1” mean there are no training time, it is undergoing only a prediction)

	accurac	precision	recall	F1_score	sensitivity	specificity	error_rate	mean_cross_val_score	auc_score	train time	pred time
AdaBoost (fine_train)	0.9722	0.9493	0.9321	0.9406	0.9321	0.9846	0.0278	0.8345	0.8681	203.6425	0.4780
AdaBoost (fine_tune_test)	0.8450	0.6786	0.6342	0.6556	0.6342	0.9089	0.1550	0.8209	0.8374	-1.0000	0.1322
AdaBoost (test)	0.8626	0.7491	0.6156	0.6758	0.6156	0.9375	0.1374	0.8602	0.9114	-1.0000	0.1960
AdaBoost (train)	0.8595	0.7543	0.6004	0.6686	0.6004	0.9396	0.1405	0.8587	0.9112	6.0628	0.6807
Artificial Neural Network (fine_train)	0.8510	0.7402	0.5683	0.6430	0.5683	0.9383	0.1490	0.8494	0.9004	3128.9858	0.0294
Artificial Neural Network (fine_tune_test)	0.8537	0.7344	0.5813	0.6490	0.5813	0.9363	0.1463	0.8514	0.9022	-1.0000	0.0239
Artificial Neural Network (test)	0.8537	0.6987	0.6524	0.6748	0.6524	0.9147	0.1463	0.8435	0.8883	-1.0000	0.0339
Artificial Neural Network (train)	0.8872	0.7856	0.7184	0.7505	0.7184	0.9394	0.1128	0.8469	0.8918	96.9095	0.1581
Decision Tree (fine_train)	0.8452	0.6878	0.6129	0.6482	0.6129	0.9157	0.1548	0.8319	0.8193	-1.0000	0.0050
Decision Tree (test)	0.8286	0.6359	0.6156	0.6256	0.6156	0.8932	0.1714	0.8100	0.7454	-1.0000	0.0070
Decision Tree (train)	0.9722	0.9803	0.9004	0.9387	0.9004	0.9944	0.0278	0.8200	0.7674	0.4952	0.0170
Decision tree (fine_train)	0.9172	0.8788	0.7530	0.8110	0.7530	0.9679	0.0828	0.8360	0.8375	2.5234	0.0155
K Nearest Neighbour (fine_train)	0.8741	0.7642	0.6750	0.7168	0.6750	0.9356	0.1259	0.8471	0.8868	167.4556	71.2494
K Nearest Neighbours (fine_tune_test)	0.8522	0.6988	0.6404	0.6684	0.6404	0.9163	0.1478	0.8419	0.8812	-1.0000	16.5723
K Nearest Neighbours (test)	0.8449	0.6749	0.6431	0.6586	0.6431	0.9061	0.1551	0.8349	0.8562	-1.0000	14.0477
K Nearest Neighbours (train)	0.8841	0.7812	0.7070	0.7423	0.7070	0.9388	0.1159	0.8381	0.8648	4.0206	52.6584
Logistic Regression (fine_train)	0.8511	0.7295	0.5869	0.6505	0.5869	0.9327	0.1489	0.8503	0.9016	51.3645	0.0060
Logistic Regression (fine_tune_test)	0.8546	0.7247	0.6049	0.6594	0.6049	0.9303	0.1454	0.8528	0.9047	-1.0000	0.0010
Logistic Regression (test)	0.8543	0.7240	0.6040	0.6586	0.6040	0.9302	0.1457	0.8528	0.9047	-1.0000	0.0010
Logistic Regression (train)	0.8510	0.7293	0.5866	0.6502	0.5866	0.9327	0.1490	0.8502	0.9016	3.5185	0.0060
Random Forest (fine_train)	0.9722	0.9571	0.9237	0.9401	0.9237	0.9872	0.0278	0.8488	0.8950	381.9646	1.2415
Random Forest (fine_tune_test)	0.8569	0.7159	0.6382	0.6748	0.6382	0.9232	0.1431	0.8471	0.8930	-1.0000	0.5286
Random Forest (test)	0.8555	0.7113	0.6373	0.6723	0.6373	0.9216	0.1445	0.8461	0.8891	-1.0000	0.3295
Random Forest (train)	0.9722	0.9576	0.9229	0.9400	0.9229	0.9874	0.0278	0.8466	0.8920	30.9596	0.8449
SGD Classifier (fine_train)	0.8364	0.8101	0.4012	0.5367	0.4012	0.9709	0.1636	0.8369	0.8883	96.9910	0.0060
SGD Classifier (fine_tune_test)	0.8413	0.8150	0.4111	0.5465	0.4111	0.9717	0.1587	0.8346	0.8933	-1.0000	0.0010
SGD Classifier (test)	0.8532	0.7503	0.5529	0.6366	0.5529	0.9442	0.1468	0.8434	0.8940	-1.0000	0.0020
SGD Classifier (train)	0.8499	0.7553	0.5386	0.6288	0.5386	0.9461	0.1501	0.8481	0.8985	12.2148	0.0060
Support Vector Machine (fine_train)	0.8529	0.7740	0.5327	0.6311	0.5327	0.9519	0.1471	0.8506	0.9019	1893.0217	49.1957
Support Vector Machine (fine_tune_test)	0.8554	0.7688	0.5409	0.6350	0.5409	0.9507	0.1446	0.8530	0.9020	-1.0000	12.7355
Support Vector Machine (test)	0.8548	0.7527	0.5600	0.6422	0.5600	0.9442	0.1452	0.8560	0.9020	-1.0000	15.5275
Support Vector Machine (train)	0.8528	0.7575	0.5540	0.6399	0.5540	0.9452	0.1472	0.8515	0.9009	98.4937	58.8210

3.2 PERFORMANCE ANALYSIS

In overall, we can classify each score as following

Accuracy	It will indirectly to show us the model is overfitting or in underfitting
Precision	The model positive trust level
Recall	Positive predicted is true
F1	Can get high Precision and high Recall in same time
TPR	Can said mostly same as Recall, the higher the better
FPR	Negative sample that predict as positive. The lower the better
AUC	Turn ROC curve to a quantitative value (Showing TPR and FPR)
Training Time	The shorter the better
Predict Time	The shorter the better

The green colour highlighted parameter will be our focus according to the objective of this project.

The recall is more important than precision. Our objective is to identify potential customers to sell our products. Thus, classifying a potential customer into non-potential group will cause loss of profit. On the other hand, classifying a non-potential customer in potential group does not matter as we do not want to miss any opportunities.

In short, if we have a high AUC value but a low F1 score, this is possible to find the threshold where F1 score is decent. However, if the F1 score is high but AUC is low, that means the classifier model is working good on the current threshold but it will not do so for other thresholds (a bad classifier). On the other hand, if both values (AUC and F1) are high, means that the classifier works well at all thresholds, in vice versa, if the both values are low, the classifier is not good and the adjustment of the threshold is useless.

For prediction time, there will not have much difference for fine-tuned and not fine-tuned (train set will usually longer then test set as train set is larger). The slight difference of the prediction time is because of the status of our computer and kernel. Lastly for AUC score, F1 score and recall score we use the score after fine tune to analyse as it has the best parameter which is obtained from random search fine tune.

3.2.1 MODEL

General guidelines for performance analysis:

Legend	Training Time	Prediction Time	The final prediction score	
			F1	Recall
Very Fast/ Good	$\leq 5s$	$\leq 0.01s$	> 0.67	> 0.64
Fast/ Above average	$> 5s$	$> 0.01s$	> 0.65	> 0.62
Medium	$> 10s$	$> 0.10s$	> 0.63	> 0.60
Slow/ Below average	$> 50s$	$> 0.50s$	> 0.61	> 0.50
Very Slow/ Low	$> 80s$	$> 1.00s$	≤ 0.61	≤ 0.50

AUC score general guidelines:

0.5 = Very poor, we might as well flip a coin.

0.5-0.7 = Poor, not much better than a coin toss.

0.7-0.8 = Acceptable

0.8-0.9 = Excellent

> 0.9 = Outstanding

This performance analysis will be mainly focusing on testing performance after fine-tuning (AUC, recall, F1) whereas the training time we use the time for normal training but not the fine tune, since the param and n iteration of each algorithm are not same:

Algorithm	Time taken		Comment
	Train	Predict	
AdaBoost	Fast	Slow	<ul style="list-style-type: none"> • The recall score is above average when compared to other algorithms. • The F1 score is above average. • Since there is no significant difference between the training accuracy and its cross-validation accuracy, we can conclude that AdaBoost is not overfitting. Both accuracies are around 85% indicating that there is no underfitting issue as well.

			<ul style="list-style-type: none"> • After evaluating the model on unseen data, the AUC score of test set has increased by 0.0002 which is slightly better than expected. • After fine-tuning, the AUC score decreased by 0.074 indicating that the performance has unexpectedly decreased. This might be due to hyperparameters found using random search were not optimal. • The AUC score in final prediction after fine-tuning is 0.8374 but the AUC score of test data before fine-tuning was 0.9114, this scenario shows that AdaBoost performance may not be stable in our datasets. Thus, it will not be a consideration for using this model.
Artificial Neural Network (ANN)	Very Slow	Very Fast	<ul style="list-style-type: none"> • The recall score is below average when compared with other algorithms. • The F1 score is average among all the other classifiers. • The AUC score is outstanding. • Since there is no significant difference between the training accuracy and its cross-validation accuracy, we can conclude that ANN is not overfitting. Both accuracies range between 84% to 89% indicates that there is no underfitting issue as well. • After evaluating the model on unseen data, the AUC score of test set only decreased by 0.0035 which can be considered good. • After fine-tuning, the AUC score increased by 0.0139 indicating that the performance slightly increased. • The AUC score in final prediction after fine-tuning is 0.9022 which can be considered good.

Decision Tree	Very Fast	Fast	<ul style="list-style-type: none"> • The recall score is average. • The F1 score is average. • The AUC score is excellent. • The train accuracy is higher than its cross-validation accuracy by only 0.1522 indicating that there is no overfitting issue. Both accuracies are above 80% which is still considered high, thus there is no underfitting issue. • After evaluating the model on unseen data, the AUC score of test set decreased by 0.022. • After fine-tuning, the AUC score increased by 0.739 indicating that the performance has increased significantly. We can conclude that fine-tuning is necessary for Decision Tree to provide better predictions. • The AUC score in final prediction after fine-tuning is 0.8193 which is above average. • One advantage is that the prediction can be done in a short time. • However, if we only have short time to train, this will not be the best algorithm. But in real world situation, have a lot of time to train our train set. • So, we can conclude that Decision tree is not suitable for us to use.
K Nearest Neighbour	Very Fast	Very Slow	<ul style="list-style-type: none"> • The recall score is average. • The F1 score is average. • The AUC score is excellent. • The train accuracy is higher than its cross-validation accuracy by 0.46 which is quite significant indicating that there is an overfitting issue. Both accuracies are above 80% which is still considered high, thus there is no underfitting issue.

			<ul style="list-style-type: none"> • After evaluating the model on unseen data, the AUC score of test set only decreased by 0.0086 which can be considered good. • After fine-tuning, the AUC score increased by 0.025 indicating that the performance has slightly increased. • The AUC score in final prediction after fine-tuning is 0.8812 which can be considered good. • But the shortage is the prediction is very slow. It is useful in a situation which need less training time but longer testing time is acceptable. Thus, in a situation which require to train and predict in short time, this model can be a good choice. • Because of the prediction time, it is not suitable as our model algorithm
Logistic Regression	Very Fast	Very Fast	<ul style="list-style-type: none"> • The recall score is medium. • The F1 score is above average. • The AUC score is outstanding. • Since there is no significant difference between the training accuracy and its cross-validation accuracy, we can conclude that it is not overfitting. Both accuracies are around 85% indicating that there is no underfitting issue as well. • After evaluating the model on unseen data, the AUC score of test set only decreased by 0.0031 which can be considered good. • After fine-tuning, there is no change in the AUC score. • The AUC score in final prediction after fine-tuning is 0.9047 which can be considered good. • If we have short time to train, this will be the best algorithm.

			<ul style="list-style-type: none"> • One advantage is that the prediction can be done in a short time. • But the AUC score is high, which mean that we can safely conclude that, this algorithm can be put inside our potential-model-list
Random Forest	Medium	Slow	<ul style="list-style-type: none"> • The recall score is above average. • F1 score is above the very good. • AUC score is excellent. • The train accuracy is higher than its cross-validation accuracy by only 0.1256 indicating that there is no overfitting issue. Both accuracies are above 80% which is still considered high, thus there is no underfitting issue. • After evaluating the model on unseen data, the AUC score of test set only decreased by 0.0029 which can be considered good. • After fine-tuning, the AUC score of test set increased by 0.0039 indicating that the performance has slightly increased. • The AUC score in final prediction after fine-tuning is 0.8930 which can be considered excellent. • If we only have short time to train, this will be the best algorithm. • One advantage is that the prediction can be done in a short time. • We can safely conclude that, this algorithm can be putted inside our potential-model-list.
SGD Classifier	Medium	Very Fast	<ul style="list-style-type: none"> • The recall score is the lowest. • F1 score is the lowest. • AUC score is excellent. • The train accuracy is higher than its cross-validation accuracy by only 0.0018 indicating that there is no

			<p>overfitting issue. Both accuracies are above 80% which is still considered high, thus there is no underfitting issue.</p> <ul style="list-style-type: none"> • After evaluating the model on unseen data, the AUC score of test set only decreased by 0.0045 which can be considered good. • After fine-tuning, the AUC score of test set decreased by 0.0007. This is probably because the hyperparameters found using random search were not optimal. However, the decrease is less than 0.001, it can be said that the hyperparameters found were close to optimal. • The AUC score in final prediction after fine-tuning is 0.8933 which can be considered good. • One advantage is that the prediction can be done in a short time. • However, the recall score is only 0.5386 which is not suitable for our model selection.
Support Vector Machine	Very Slow	Very Slow	<ul style="list-style-type: none"> • The recall score is low. • The F1 score is above average. • The AUC score is outstanding. • Since there is no significant difference between the training accuracy and its cross-validation accuracy, we can conclude that it is not overfitting. Both accuracies are around 85% indicating that there is no underfitting issue as well. • After evaluating the model on unseen data, the AUC score of test set has increased by 0.0011. The test AUC is higher than train AUC which is better than expected. • After fine-tuning, there is no change in the AUC score.

			<ul style="list-style-type: none"> • The AUC score in final prediction after fine-tuning is 0.9020 which can be considered good. • This algorithm can be said that is the worst algorithm when compared to other in terms of training and prediction time. Therefore, it will not be put into consideration for model selection.
--	--	--	--

In summation, ANN, Logistic Regression and Random Forest are potentially good classifiers in our project. The best model will be determined in the later section – [3.3 Model Comparison](#).

3.2.2 FINE TUNE ANALYSIS

Following is the results we obtained. We use AdaBoost Algorithm as example.

AdaBoost Algorithm	accuracy	precision	recall	F1_score	sensitivity	specificity	error_rate	mean_cross_val_score	auc_score	train time	pred time
Grid Search	0.9722	0.9545	0.9264	0.9403	0.9264	0.9864	0.0278	0.8366	0.8714	306.1463	0.871
Random Search	0.9722	0.9493	0.9321	0.9406	0.9321	0.9846	0.0278	0.8345	0.8681	203.6425	0.478

As mentioned in [2.3.3 Tuning and Testing](#), we will use random search as our fine tune method. Here is our comparison. To be have only one manipulated variable, we set the iteration for random search is 9 as the combination of grid search is $3 * 3 = 9$.

We choose the AdaBoost algorithm as our testing algorithm. Random Search and Grid Search is completed in around same timing (The time will slightly change in each run due to performance of hardware, and might be faster in a good GPU).

But as we can see, a random search can put a more extensive range of hyperparameter as it only randomly selected the subset, while grid search is limited because grid search needs to test all combination. Thus, if we set a higher range of the number of parameters on grid search, the time consuming will be increased in exponentially.

In short, random search fine tune can give a higher AUC score, recall score, and F1 score in a short time compared to grid search.

3.2.3 OUTLIER ANALYSIS

The figure below shows the result before we implement removal of outlier rows.

AUC score	Before	After
AdaBoost (fine_tune_test)	0.8113	0.8374
Artificial Neural Network (fine_tune_test)	0.8901	0.9022
Decision Tree (fine_tune_test)	0.7747	0.8193
K Nearest Neighbours (fine_tune_test)	0.8353	0.8812
Logistic Regression (fine_tune_test)	0.9036	0.9047
Random Forest (fine_tune_test)	0.8904	0.8930
SGD Classifier (fine_tune_test)	0.8271	0.8933
Support Vector Machine (fine_tune_test)	0.8921	0.9020

The outlier removal is work in every training model. As we can see outlier removal is very important to increase our AUC score.

3.3 MODEL COMPARISON

We will choose Random Forest as our primary choice of algorithm model. As in the dataset, we can see that the distribution of income. We can consider that about 1/5 population of the city is rich and have more potential to become a buyer for household, car and for some expensive entertainment activity such as golf, etc. Thus, in order to classify them accurate, our model chosen must be a high True Positive Rate (TPR) and high recall score and high auc.

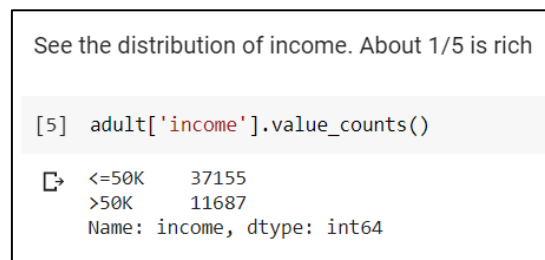


Figure 3.4.1 The distribution of income of population of the city

On the other hand, the precision score is not as important as the recall score, as it will only lead us to promote an advertisement to low-income people (non our target customer). This type of error is acceptable, they can be a platform to further advertise our product. Moreover, they maybe can be our buyer too, just they have lower potential as than the more income buyer. If we classify a rich man to be poor, then will lose a potential buyer.

As shown in [3.1.2 Best model in all Dataset](#), Random Forest is the chosen algorithm. It is fast in training and prediction. Moreover, it is no underfitting and overfitting, and above average F1 score, a high recall value and having a high AUC score.

When we compare the AUC score after fine tuning, no other classifiers have the higher AUC score than the Logistic Regression, Only ANN, K Nearest Neighbour (KNN), Random Forest, SGD and SVM has a quite close AUC score compare to the Logistic Regression. Random Forest AUC score only have a slight difference with Logistic Regression. The classifier with the lowest AUC is the Decision Tree.

When comparing the recall/TPR at certain threshold, KNN has the highest after fine tuning and testing out on the unseen data, follow by Random forest. When comparing our chosen model, Random Forest to these classifiers, we found that the Random Forest recall/TPR is the highest among the model-based algorithms. The classifier with the worst recall is the SGD classifier with only (0.4111).

When comparing in term of accuracy, the highest accuracy is obtained by the SGD classifier after fine tuning, however SGD classifier has a really low recall/ TPR score. The average of other classifiers is around 0.63 to 0.74. From the table we can see that the precision of Random Forest is quite good (0.72) when comparing to other classifiers while having the highest recall/TPR. The classifier with the lowest precision is the Decision Tree.

Furthermore, when comparing the F1 score, Random Forest has the highest follow by KNN. This mean that at certain threshold, Random Forest can get a better and acceptable precession and while have a high recall. However, Logistic Regression is once again, only slightly behind them but have more AUC score then them. The classifier with the lowest F1 score is the SGD classifier.

Logistic Regression has the fastest prediction time among other algorithm, the only one algorithm that has same prediction time with it is the SGD classifier. But the SGD classifier has the worst recall and average AUC compare to all other algorithm. When we add in these 2 elements, recall/TPR and AUC, to find the best model. KNN is not suitable to be the selected choice as it requires large amount of prediction time comparing to other model-based algorithm, although it has high recall, but it has a slow prediction time in real world is not acceptable, thus it is not chosen.

When comparing the prediction time of Random Forest to Logistic Regression, Random Forest is slower than it, the different is about 0.5 second. Random Forest is actually more suitable to be the selected choice as it has a high recall which is what we want, and high F1 than the Logistic Regression but lower AUC and a slower prediction time. However, in our case, we are able to tolerate to the slower prediction time as it is just under 1 second of difference.

In short, combining the important performance measure for our goal, the best model among these classifiers is Random Forest as we can tolerate to it lower specificity, precision, but because it has a good F1 score we can still get an acceptable precision while have the highest recall.

Chapter 4. CONCLUSION

This report proposed Random Forest to be the best classification model in this project. After fine-tuning, the AUC score obtained by Random Forest was 0.8930 which contributes to excellent performance. Although the AUC score of Random Forest is not the highest on list, when we take other metrics which are recall and F1 score into considerations, Random Forest is the most suitable model in this project in order to achieve our goal – to identify all the potential customers. The future scope of this project is to determine the interactions amongst the features using automated methods. By determining the importance of features, the performance can be further improved using dimensionality reduction techniques such as Principal Component Analysis (PCA).