IAT – Machine Learning Algorithms

Project Assignment

Siwoo JUNG

Table of Contents

1. Project Scope

The task is to help an organization better understand their customers' needs and increase customer loyalty by performing customer segmentation. Given a customer data, including various information, such as gender, age, income, spending score, customer ID and whether a customer has bought a product or not, three machine learning algorithms are performed – supervised, unsupervised, and semi-supervised. In addition, results of each algorithm are analyzed and compared.

In this report, the machine learning algorithms include random forest for supervised, K-means clustering for unsupervised and label propagation for semi-supervised. However, regardless of the algorithms, each model should predict a value of a column 'Buy', whose data is represented in binary. To do so, the overall sequence begins with data exploration, such as exploratory data analysis, followed by data pre-processing, model implementation, optimization, and evaluation.

In business perspective, the machine learning models will guide which variables are particularly more important to understand their customers' loyalty. Therefore, the company can adjust their business strategy as per those priority variables. Furthermore, the company could evaluate the needs and loyalty of future customer with the models and future customer data.

2. Attachment

Along with the report, the raw .ipynb Python code file, named 'code_Siwoo_JUNG', has been attached. This single file includes all the raw codes done for this project.

3. Data Exploration

The first procedure is exploring data, and it often entails Exploratory Data Analysis (EDA). EDA is a technique to understand overall characteristics of data, such as patterns and anomalies, by analyzing and investigating data with the aid of data visualization. EDA techniques generally include univariate, bivariate, multivariate visualization, and dimensionality reduction. In this project, univariate and bivariate methods are exclusively applied.

3.1 Data preparation

As shown below, the original data includes 200 rows of customer data, such as ID, gender, age, annual income, spending score and an indicator whether a customer has purchased, 'Buy'. However, data types of age and annual income are initially string; some of them even have space characters. Therefore, for effective data exploration, space characters were replaced to NaN and all the numbers in string were converted to float. For a column 'Gender', Male was replaced to 0 and Female was replaced to 1, as it is described in binary. Most importantly, the column 'CustomerID', was initially dropped as it is definitely not a contributing factor, and for efficient data exploration. For convenience, the column name of 'Annual Income (k$)' and 'Spending Score (1-100)' were replaced to 'AnnualIncome' and 'SpendingScore', respectively.

In [ ]:
```
# Load dataset
db = pd.read_csv('dataset.csv')
db.head()
```

Out[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Buy
0	1	Male	19	15	39	0
1	2	Male		15	81	1
2	3	Female	20	16	6	0
3	4	Female	23	16	77	0
4	5	Male	31	17	40	1

```
In [ ]: db.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Gender         200 non-null    int64
 1   Age            200 non-null    object
 2   AnnualIncome   200 non-null    object
 3   SpendingScore  200 non-null    int64
 4   Buy            200 non-null    int64
dtypes: int64(3), object(2)
memory usage: 7.9+ KB
```

3.2 Actual data exploration

3.2.1 Missing data

To begin with, any existing missing data was analyzed. As discussed, columns 'Age' and 'AnnualIncome' initially had space characters. Therefore, it was observed that these columns included missing data, as shown in Figure 1.

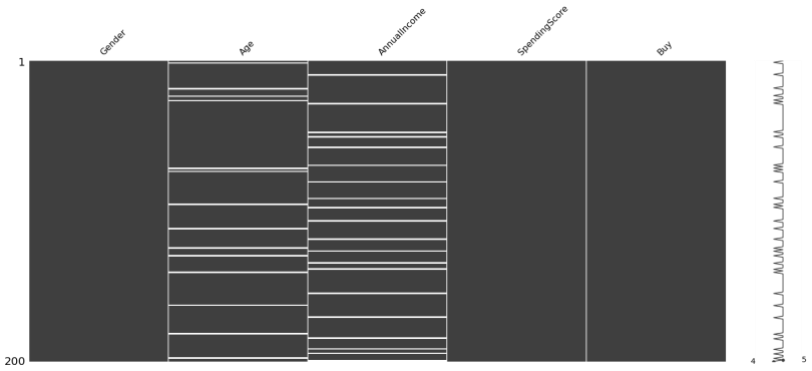

Figure 1. Visualisation of missing data

3.2.2 Correlation

Correlation of variables shows how variables are related to each other, either positively or negatively. As shown in heatmap in Figure 2, it is noticeable that the variable 'Buy' is highly and positively correlated to spending score of a customer, yet it is merely and negatively.

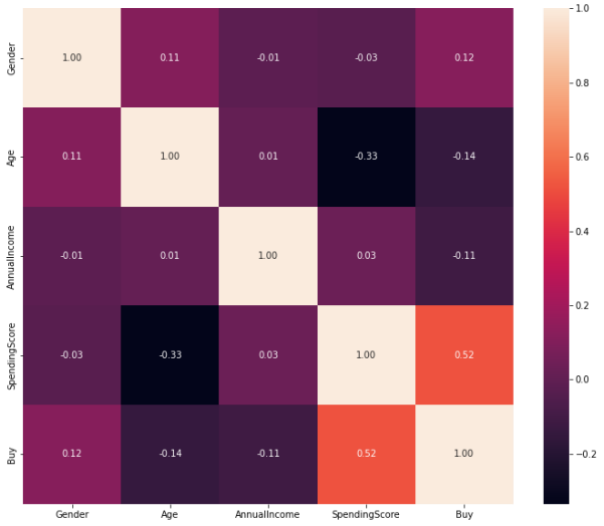

Figure 2. Correlation heatmap

3.2.3 Histogram

Figure 3 shows distribution of each variable. It is noticed that a number of female is slightly higher than that of male. Ages and annual incomes of customers are positively skewed. Spending score of customers shows normal distribution, yet not perfect. Regarding 'Buy', there is a considerably higher number of customers who have not bought.

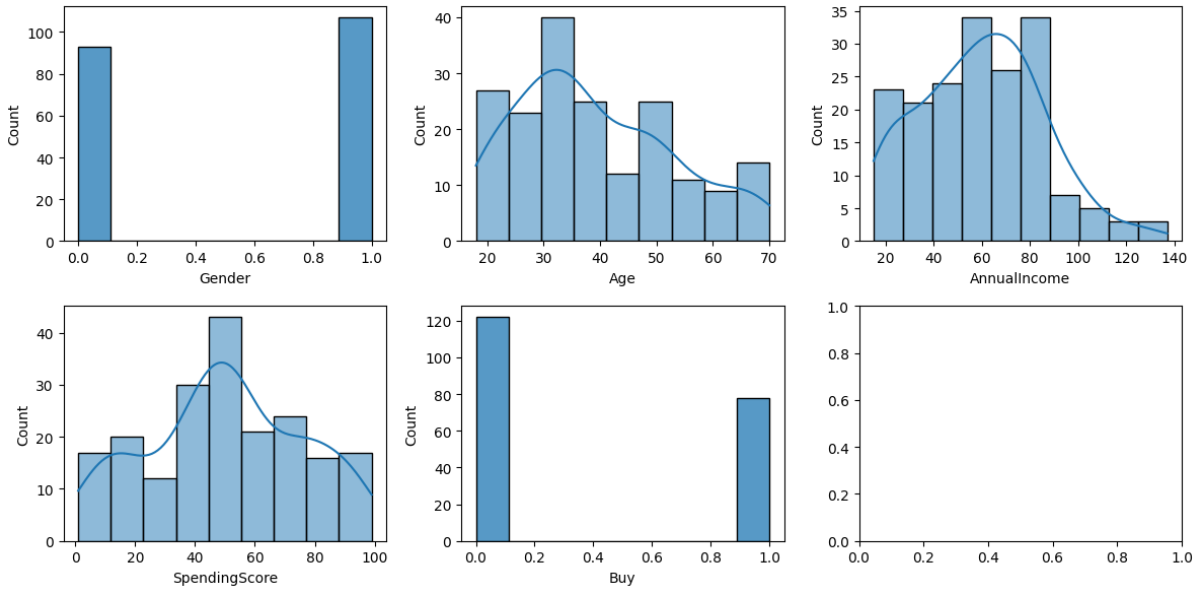

Figure 3. Distribution of each variable

3.2.4 Pairplot

Pairplot shows distribution and relationship between two datasets, as shown in Figure 4. It is noticeable that there are no meaningful relationships among age, annual income and spending score.

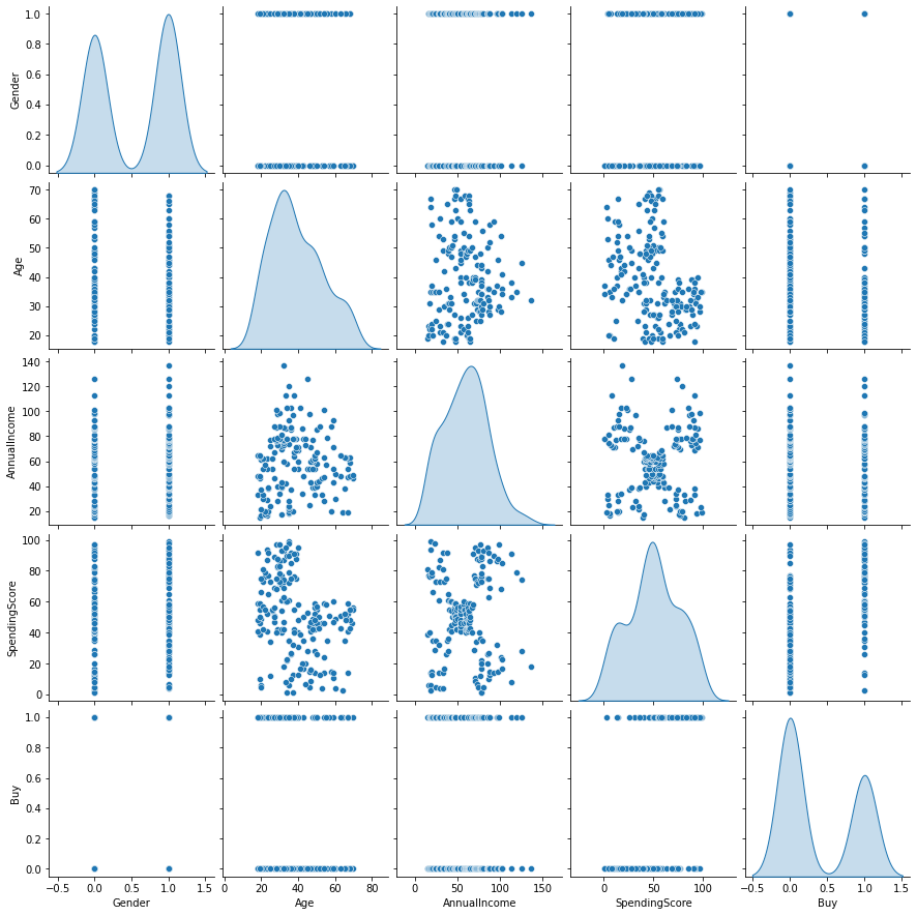

Figure 4. Pairplot of variables

3.2.5 Boxplot

Boxplot shows several information of each variable, such as minimum, lower quartile, median, upper quartile, maximum and outliers. Figure 5 shows that there is one outlier from annual income; however, annual income of a person at approximately $140,000 is a feasible and acceptable, so it was kept throughout the project.

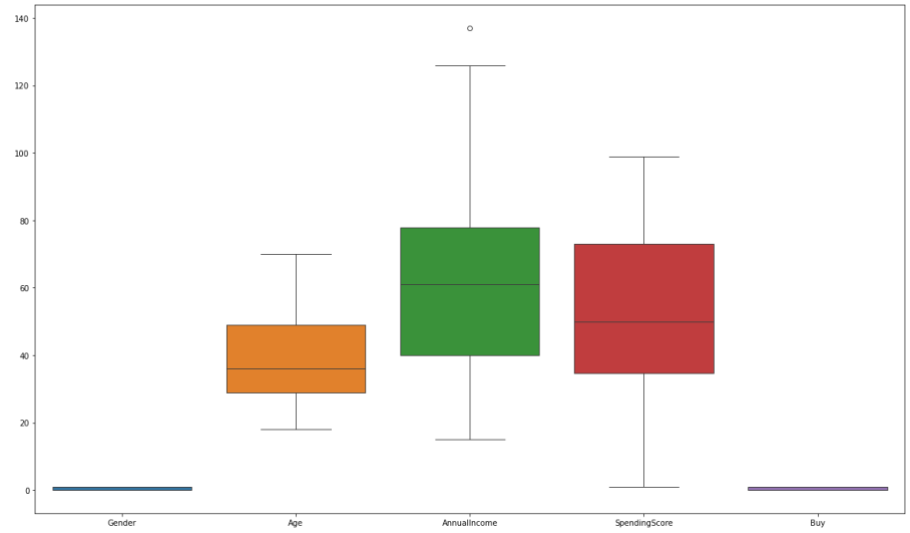

Figure 5. Boxplot of variables

3.3 Pre-processing

Most of pre-processing was described in **3.1.** However, missing data from age and annual income had to be handled. For these variables, NaN items were replaced to mean values of each variable.

```
# Imputation
db['Age'].fillna(db['Age'].mean(), inplace=True)
db['AnnualIncome'].fillna(db['AnnualIncome'].mean(), inplace=True)

print((db.isnull()).sum())
```

```
Gender           0
Age              0
AnnualIncome     0
SpendingScore    0
Buy              0
dtype: int64
```

4. Implementation of Machine Learning Algorithms

In this section, implementation of machine learning algorithms of supervised, unsupervised, and semi-supervised are discussed. For the better structure of the report, optimization algorithms, which is supposed to be part 4 as per the project guideline, is also instead addressed in this section.

4.1 Supervised (Random Forest)

4.1.1 Model selection

Supervised machine learning algorithms use labelled data to train the model. It is separated into two main types, regression, and classification, such as linear regression, logistic regression, support vector machines, random forest, and so on. In this problem, the dependent variable is in binary, which leads to a choice of logistic regression, which is used to solve binary classification problems. However, logistic regression assumes linear relationship between independent and dependent variables, whereas actual relationship between independent and dependent variables in this problem is rather non-linear, as shown in Figure 2 and Figure 4. Therefore, random forest was considered in this project, which creates various decision trees to determine a single result and is flexible to both linear and non-linear data.

4.1.2 Training

First, a dependent variable, which is the column ‘Buy’, was declared as ‘y’, and other independent variables were declared as ‘X’.

```
In [ ]: X = db.drop(['Buy'], axis = 1)
        y = db['Buy']
```

Those variables then were split into training and testing sets, where training size is 80% and testing size is 20%. Considering there is a noticeable difference in the number of 0 and 1 in the dependent variable, as shown in Figure 1, stratified split was considered.

```
In [ ]: # Split the dataset into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=21, stratify=y)
```

Dependent variables were then normalized. A standard normalization, or Z-score normalization, was applied.

```
In [ ]: # Apply normalisation
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```

With the scikit-learning’s random forest classifier method, data was trained. As it is an initial attempt, default hyperparameters were applied.

```
In [ ]: # create a random forest classifier
        rf = RandomForestClassifier()

        # fit the random forest on the training dataset
        rf.fit(X_train, y_train)

        # Make predictions on the test set
        y_pred = rf.predict(X_test)
```

4.1.3 Evaluation

As random forest is a classification method, several evaluation metrics, such as accuracy, precision, recall, F1, and ROC-AUC, were used. To test whether the model was underfitting or overfitting, the accuracy of the train set and the test set was compared. The model, without any hyperparameter tuned, was overfitting as shown below.

```
In [ ]: # create a random forest classifier
        rf = RandomForestClassifier()

        # fit the random forest on the training dataset
        rf.fit(X_train, y_train)

        # Make predictions on the test set
        y_pred = rf.predict(X_test)

        # Calculate and print the accuracy score
        train_accuracy = accuracy_score(y_train, rf.predict(X_train))
        print(f"Train Accuracy: {train_accuracy:.2f}")

        accuracy = accuracy_score(y_test, y_pred)
        print(f"Test Accuracy: {accuracy:.2f}")

Train Accuracy: 1.00
Test Accuracy: 0.82
```

As shown below and Figure 6, the overall accuracy is 0.82, indicating that the model could determine whether a customer has bought a product or not. Precision is 0.76, meaning that among those predicted to have purchased a product, actually 76% has truly purchased a product. 0.81 of recall means that the model can determine 81% of all customers who have actually purchased a product. On the other hand, F1 is a balanced metric between precision and recall; therefore, the model performs overall 79% correctness.

```
In [ ]: # Evaluation metrics
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        print(f"Accuracy: {accuracy:.2f}")
        print(f"Precision: {precision:.2f}")
        print(f"Recall: {recall:.2f}")
        print(f"F1: {f1:.2f}")

Accuracy: 0.82
Precision: 0.76
Recall: 0.81
F1: 0.79
```

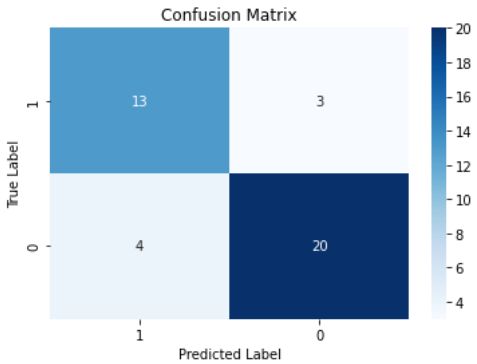

Figure 6. Confusion matrix of initial random forest classification

Figure 7 shows AUC-ROC score of 0.90, where the model performance is generally acceptable. Considering imbalance of 'Buy' from original data, AUC-ROC score is more suitable than accuracy. However, the model was overfitting, and considering the objective of the problem, higher recall is desired.

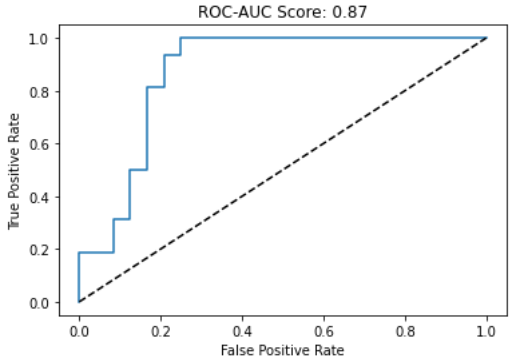

Figure 7. ROC of initial random forest classification

4.1.4 Optimisation of algorithms

To address the overfitting issue and lower recall, it was decided to optimise the model by tuning hyperparameters. RandomFroestClassifier from scikit-learning has several hyperparameters, such as 'max_depth', 'max_features', 'min_samples_leaf' and 'n_estimators'.

4.1.4.1 Using GridSearchCV

Scikit-learning also offers GridSearchCV, which determines the optimal variables by tuning. As shown below, by using GridSearchCV, optimised hyperparameters were determined with the best accuracy of 0.78.

```
In [ ]: # Set parameter grid
params = {'bootstrap': [True],
          'max_depth': [4, 6, 8],
          'max_features': [2, 4, 6],
          'min_samples_leaf': [2, 4, 6],
          'n_estimators': [10, 100, 1000]}

# Apply GridSearchCV
rfc = RandomForestClassifier(random_state = 0, n_jobs = -1)
grid_cv = GridSearchCV(rfc, param_grid = params, n_jobs = -1, cv=StratifiedKFold(), scoring='recall')
grid_cv.fit(X_train, y_train)

print('Optimised hyper params: ', grid_cv.best_params_)
print(f'Best accuracy: {grid_cv.best_score_:.2f}')
```

```
Optimised hyper params:  {'bootstrap': True, 'max_depth': 4, 'max_features': 2, 'min_samples_leaf': 6, 'n_estimators': 1000}
Best accuracy: 0.78
```

A new model was built with the same optimized hyperparameters and was trained with the same procedures. Train and test accuracies were 0.81 and 0.82, respectively, which resolved the overfitting issue.

```
In [ ]: # Build a new model with optimised hyper parameters retrieved above
rf_optimised = RandomForestClassifier(n_estimators = 1000,
                                      max_depth = 4,
                                      max_features = 2,
                                      min_samples_leaf = 6,
                                      random_state = 0,
                                      bootstrap=True,
                                      n_jobs = -1)
rf_optimised.fit(X_train, y_train)
y_pred_train = rf_optimised.predict(X_train)
print(f'Train accuracy: {accuracy_score(y_train,y_pred_train):.2f}')
y_pred = rf_optimised.predict(X_test)
print(f'Test accuracy: {accuracy_score(y_test,y_pred):.2f}')
```

```
Train accuracy: 0.81
Test accuracy: 0.82
```

Also, all of accuracy, precision, recall and F1 improved, compared to those before tuning hyperparameters.

```
In [ ]: accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1: {f1:.2f}")
```

```
Accuracy: 0.82
Precision: 0.76
Recall: 0.81
F1: 0.79
```

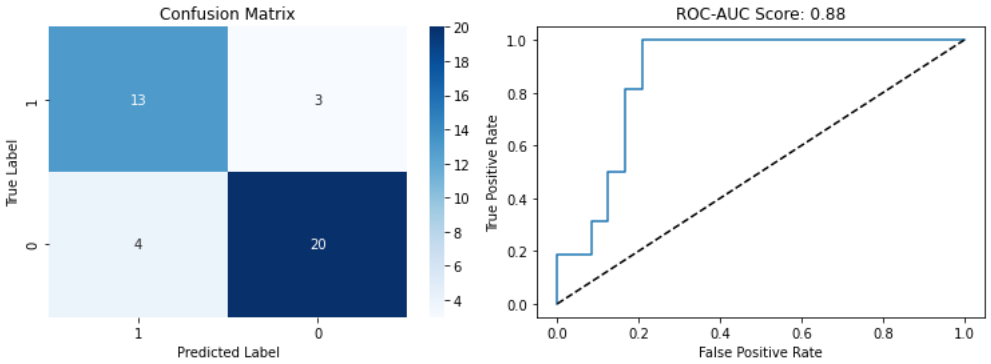

Figure 8. Confusion matrix (left) and ROC (right) after tuning hyperparameters using GridSearchCV

4.1.4.2 Manually tuning hyperparameters

However, for the successful business, better estimation of customer loyalty was desired. Therefore, another tuning of hyperparameters was done by manually tuning hyperparameters. By applying follow hyperparameters after trial-and-error, the highest evaluation metrics was achieved.

```
# create a random forest classifier
rf = RandomForestClassifier(n_estimators=100, max_depth=4, max_features=5,
                            min_samples_leaf=5,min_samples_split=5,random_state=21)
```

While being neither overfitting nor underfitting, the model achieved improve train and test accuracy values.

```
# Calculate and print the accuracy score
train_accuracy = accuracy_score(y_train, rf.predict(X_train))
print(f"Train Accuracy: {train_accuracy:.2f}")

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Train Accuracy: 0.86
Test Accuracy: 0.88
```

As shown below and Figure 9, accuracy, precision, recall and F1 are 0.88, 0.79, 0.94 and 0.86, respectively. Especially, recall significantly improved, which means that the model could determine 94% of all customers that have purchased a product. Precision also slightly improved, proving the model is more suitable to be used for adjusting business strategy.

```
In [ ]: accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        print(f"Accuracy: {accuracy:.2f}")
        print(f"Precision: {precision:.2f}")
        print(f"Recall: {recall:.2f}")
        print(f"F1: {f1:.2f}")

        Accuracy: 0.88
        Precision: 0.79
        Recall: 0.94
        F1: 0.86
```

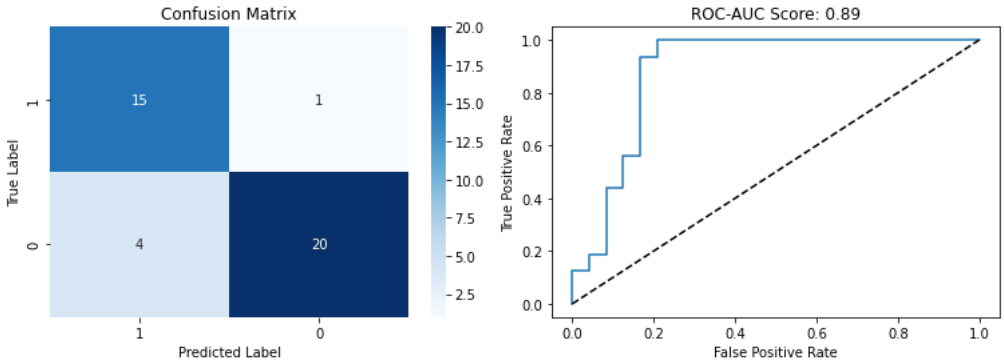

Figure 9. Confusion matrix (left) and ROC (right) after manually tuning hyperparameters

4.2 Unsupervised (K-means clustering)

4.2.1 Model selection

Unsupervised machine learning algorithm deals with unlabeled data by finding certain patterns, similarities, or differences within data without human interactions. Unsupervised algorithm includes clustering and association. Clustering is a technique that groups data based on similarities and differences, whereas association groups data based on certain rules and finds concurrence and correlation between data within a group. In this problem, K-means clustering method is suitable, which is one of an exclusive clustering technique, meaning data belongs to only one cluster. K-means clustering separates data into a certain number group based on the distance between a centroid of a group and data within a group. Considering the model should predict whether a customer has bought (or will likely buy) a product, which is described in binary and not overlapping, K-means clustering is an acceptable choice for unsupervised learning.

4.2.2 Data preparation

Due to characteristics of K-means clustering, two variables were selected. From the correlation heatmap in Figure 2, variable 'Buy' is comparably highly correlated to spending score, which is then comparably highly correlated to age of a customer. Therefore, 'Age' and 'SpendingScore' were extracted, followed by min-max normalization.

```
In [ ]: # Use Age and SpendingScore only
        X_UL = db[['Age','SpendingScore']]

        # Apply normalization
        X_UL_scale = minmax_scale(X_UL)
```

The scatterplot between age and spending score is as follow.

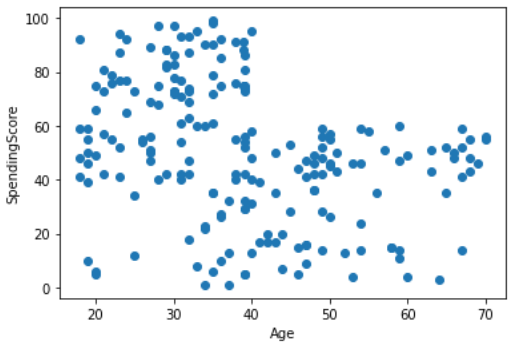

Figure 10. Scatterplot between age and spending score

4.2.3 Learning

Scikit-learning's KMeans was used to build a model. A key hyperparameter, namely 'n_clusters', which indicates number of clusters, was initially set 2. A low number of cluster could lead to underfitting and a higher number could lead to overfitting. In this problem, this number cannot be changed, as the target, the column 'Buy', is in binary.

```
In [ ]: # Due to binary classification, use 2 clsuters
kmeans = KMeans(n_clusters=2)

# Fit the input data
kmeans.fit_predict(X_UL_scale)
```

4.2.4 Evaluation

Figure 11 shows the scatterplot of clustered data. It can be noticed that data has been labelled in binary, either 0 or 1.

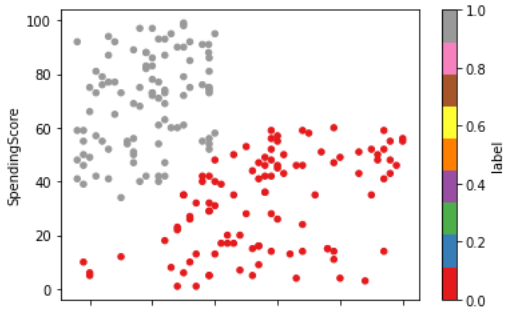

Figure 11. Visualisation of clustered data

For the metrics, same as supervised learning, accuracy, precision, recall, F1 and ROC-AUC score were compared. As shown below, accuracy, precision, recall and F1 are 0.69, 0.58, 0.58 and 0.64, respectively.

```
In [ ]: # Evaluation metrics
UL_accuracy = accuracy_score(db_UL1['Buy'], db_UL1['label'])
UL_precision = precision_score(db_UL1['Buy'], db_UL1['label'])
UL_recall = recall_score(db_UL1['Buy'], db_UL1['label'])
UL_f1 = f1_score(db_UL1['Buy'], db_UL1['label'])
print(f"Accuracy: {UL_accuracy:.2f}")
print(f"Precision: {UL_precision:.2f}")
print(f"Recall: {UL_recall:.2f}")
print(f"F1: {UL_f1:.2f}")

Accuracy: 0.69
Precision: 0.58
Recall: 0.72
F1: 0.64
```

Figure 12 shows confusion matrix and ROC. Overall, the performance of the model with K-means clustering is not effective; the model cannot accurately determine a customer who has purchased a product, which is detrimental to business strategy.

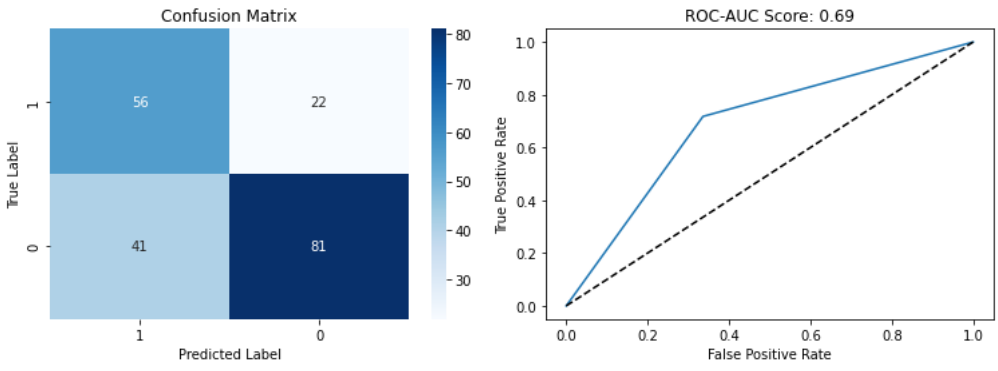

Figure 12. Confusion matrix (left) and ROC (right) with K-means clustering algorithm

4.2.5 Optimization of algorithm

Similar to the supervised learning, GridSearchCV was applied to the K-means clustering model. Generally, the number of clusters is tuned; however, as discussed in **4.2.3**, it cannot be changed. Therefore, other hyperparameters, such as 'init', 'n_init', 'max_iter', and 'algorithm' were tuned.

```
In [ ]: # Set parameter grid
params = {'init': ['k-means++', 'random'],
        'n_init':[3, 10, 100, 1000],
        'max_iter':[3, 10, 100, 1000],
        'algorithm':['lloyd', 'elkan', 'auto' 'full']}

# Apply GridSearchCV
kmc = KMeans(random_state = 0)
grid_cv = GridSearchCV(kmc, param_grid = params, n_jobs = -1)
grid_cv.fit(X_UL_scale, db['Buy'])

print('Optimised hyper params: ', grid_cv.best_params_)
print(f'Best accuracy: {grid_cv.best_score_:.2f}')
```

```
Optimised hyper params:  {'algorithm': 'lloyd', 'init': 'random', 'max_iter': 10, 'n_init': 3}
Best accuracy: -0.82
```

Although the optimized parameters were applied, all the metrics remained same. The number of clusters is the key hyperparameter, however, it could not be tuned. Otherwise, there would be multiple numbers higher than 1, which then cannot be compared to the values of the variable 'Buy'. This indicates that unsupervised learning was not a suitable model in the first place.

```
In [ ]: # Evaluation metrics
# The values are same with those before optimization

UL_accuracy2 = accuracy_score(db_UL2['Buy'], db_UL2['label'])
UL_precision2 = precision_score(db_UL2['Buy'], db_UL2['label'])
UL_recall2 = recall_score(db_UL2['Buy'], db_UL2['label'])
UL_f12 = f1_score(db_UL2['Buy'], db_UL2['label'])
print(f"Accuracy: {UL_accuracy2:.2f}")
print(f"Precision: {UL_precision2:.2f}")
print(f"Recall: {UL_recall2:.2f}")
print(f"F1: {UL_f12:.2f}")
```

```
Accuracy: 0.69
Precision: 0.58
Recall: 0.72
F1: 0.64
```

4.3 Semi-supervised

4.3.1 Model selection

Semi-supervised algorithm is a combined algorithm of supervised and unsupervised learnings, where it predicts a large portion of unlabeled data based on a smaller portion of labeled data. It includes label propagation and label spreading methods, where label propagation is hard clamping and label spreading soft clamping. Therefore, in this problem, label propagation was chosen.

4.3.2 Data preparation

Since the dependent variable is all labeled, let 70% of them unlabeled.

```
In [ ]: rand= np.random.RandomState(0)

lp = LabelPropagation()

#Defining 70% of the samples to be unlabled
random_unlabeled_samples = rand.rand(len(y)) <= 0.7

#Seperating labeled and unlabled datasets
un_labeled = np.copy(y)
un_labeled[random_unlabeled_samples] = -1
un_labeled
```

4.3.3 Learning

The model was trained using scikit-learning's LabelPropagation.

```
In [ ]: #fit the model
lp.fit(X, un_labeled)

# Predict the Labels
pred_lb = lp.predict(X)
```

4.3.4 Evaluation

Below and Figure 13 show that the performance of the model was not remarkable. Especially, the recall was significantly low that it cannot even classify half of actual loyal customers.

```
# Evaluation metrics
ls_accuracy = accuracy_score(y, pred_lb)
ls_precision = precision_score(y, pred_lb)
ls_recall = recall_score(y, pred_lb)
ls_f1 = f1_score(y, pred_lb)
print(f"Accuracy: {ls_accuracy:.2f}")
print(f"Precision: {ls_precision:.2f}")
print(f"Recall: {ls_recall:.2f}")
print(f"F1: {ls_f1:.2f}")
```

```
Accuracy of Label Spreading:  100.00%
Accuracy: 0.72
Precision: 0.73
Recall: 0.46
F1: 0.57
```

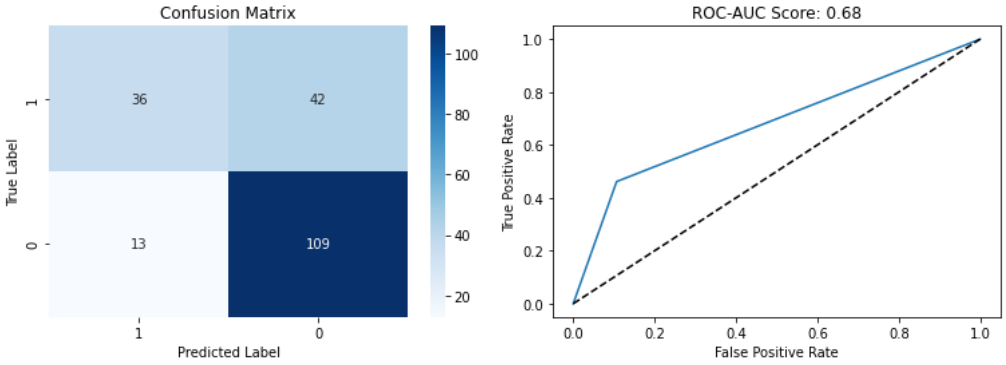

Figure 13. Confusion matrix(left) and ROC(right) by semi-supervised learning

4.3.5 Optimization of algorithm

Similar to the supervised and unsupervised learning, GridSearchCV was applied to the semi-supervised model. Hyperparameters, such as 'n_neighbors' and 'max_iter' were tuned.

```
In [ ]: # Set parameter grid
params = {'kernel': ['knn'],
        'n_neighbors': np.linspace(10,100, num=10, dtype=int),
        'max_iter': np.linspace(5,10, num=5, dtype=int)}

# Apply GridSearchCV
lp = LabelPropagation(n_jobs = -1)
grid_cv = GridSearchCV(lp, param_grid = params, scoring='recall')
grid_cv.fit(X, y)

print('Optimised hyper params: ', grid_cv.best_params_)
print(f'Best accuracy: {grid_cv.best_score_:.2f}')
```

```
Optimised hyper params:  {'kernel': 'knn', 'max_iter': 5, 'n_neighbors': 20}
Best accuracy: 0.62
```

The model was re-trained with the optimized hyperparameters. Evaluation metrics and confusion matrix after applying tuned hyperparameters are as follows. Although there is noticeable improvement, the model was not yet acceptable. With such precision and recall, the model will not effectively classify loyal customers.

```
# Evaluation metrics
ls_accuracy = accuracy_score(y, pred_lb)
ls_precision = precision_score(y, pred_lb)
ls_recall = recall_score(y, pred_lb)
ls_f1 = f1_score(y, pred_lb)
print(f"Accuracy: {ls_accuracy:.2f}")
print(f"Precision: {ls_precision:.2f}")
print(f"Recall: {ls_recall:.2f}")
print(f"F1: {ls_f1:.2f}")
```

```
Accuracy of Label Spreading:  100.00%
Accuracy: 0.73
Precision: 0.64
Recall: 0.72
F1: 0.68
```

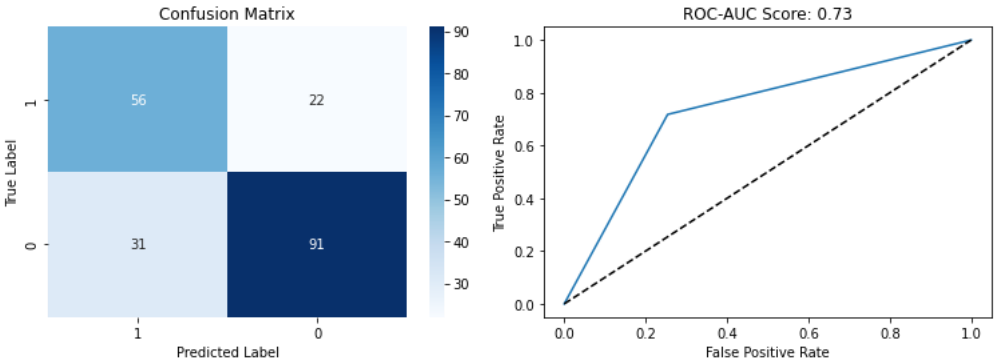

Figure 14. Confusion matrix(left) and ROC(right) by semi-supervised learning after tuning hyperparameters

5. Final Evaluation

The main task was to develop a model with various machine learning algorithms to classify and predict loyal customers. With given customer data, supervised, unsupervised and semi-supervised models were analysed and compared. For each respective algorithm, random forest, K-means clustering, and label propagation were chosen, due to various reasons discussed throughout the report.

For all models, evaluation metrics, such as accuracy, precision, recall, F1 and ROC-AUC, were applied. Considering the problem, precision and recall were comparably more important. As shown in Figure 15, random forest model outperformed in all metrics, followed by semi-supervised and unsupervised models. Considering the nature of each algorithm and the problem, supervised learning was expected to perform best in the first place. With the random forest model, the company will effectively distinguish potential loyal customers, and can plan business strategies accordingly.

This final project delivers the idea that an appropriate algorithm and model should be applied to various scenarios. Although supervised learning performed the best in this scenario, it might not be the best case, or even cannot be applied at all, in some other scenarios. Along with proper processing of data, machine-learning is a powerful tool to analyse and evaluate existing and even potential data.

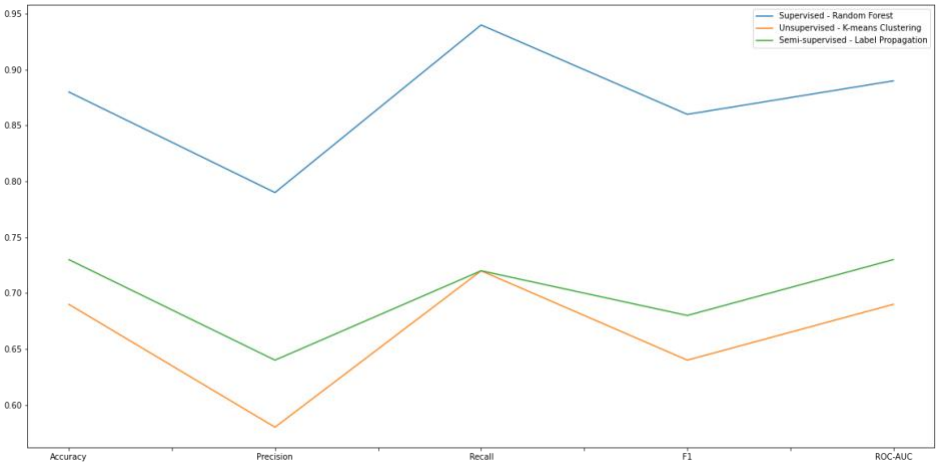

Figure 15. Comparison of evaluation metrics by different machine learning algorithms