ASSESSMENT TASK 3: DATA MINING IN ACTION

31250 Introduction to Data Analytics

MAY 19, 2025

NAME: JESSICA NGUYEN
STUDENT ID: 13289576

Table of Contents

1) Introduction 2
1.1. Data Mining Problem 2
1.2. Objective 2
1.3. Input to 'Data Mining Problem' 2
1.4. Output to 'Data Mining Problem' 2
2) Data Preprocessing 3
2.1. Data Cleaning 3
2.2. Normalisation 4
2.3. Data Transformation 5
2.4. Linear Correlation between numeric and 'subscribed' attributes 7
3) Problem-Solving Approach – Part 1: Data Mining Preparation 10
3.1 Feature Selection 10
3.2 Parameter Optimisation 10
3.3 Handling Unbalanced Data 11
4) Problem-Solving Approach – Part 2: Classification Implementation 12
4.1 K-Nearest Neighbour (KNN) 12
4.2 Decision Trees 16
4.3 Random Forest 20
4.4 Support Vector Machine (SVM) 24
4.5 Neural Network 28
4.6. Evaluation 31
5) Kaggle Submission Score of all models 32
5.1. Best model based on Kaggle Score 32
Appendix 33

1) Introduction

1.1. Data Mining Problem

Today, many individuals face the challenges on spending money in any circumstance, such as purchasing a new home, or travelling overseas or even owning any type of bank account, like a term deposit. With the current economic market on its increased cost of living, financial institutions, such as banks require exceptional strategic marketing tactics to advertise their services and products, and most importantly gain profitability. Customer types, include everyday consumers or business enterprises may think the idea of a term deposit has not much advantage to their financial situation.

Additionally, this enables for further opportunities for financial institutions to build and retain customer relationships on their products and services, specifically with term deposits. Hence, the marketing dataset has evaluated the values of the 'subscribed' attribute for a term deposit campaign based an individuals' identity, such as their job background, current financial status and their participation with the previous campaign.

1.2. Objective

The purpose of this assignment is to predict whether a person will subscribe to a term deposit campaign based on the Marketing Dataset provided. This is achieved by applying machine learning techniques that involves building and evaluating multiple classification models; K-Nearest Neighbours (KNN), Decision Trees (DT), Random Forest (RF), Support Vector Machine (SVM) and Neural Networks (NN). Finally, these classification models were put to a Kaggle assessment to determine the best score for the assignment purpose.

1.3. Input to 'Data Mining Problem'

The given Marketing Dataset has recorded a person's identity through several aspects. First is demographic; their age, job position, marital status, education, owns any housing/personal loan. Secondly, the outcomes of the previous marketing campaign; their contact type, duration and period, as well as economic indicators that evaluate their financial status. Lastly, the 'subscribed' attribute is classified as a predictor for training the models in identifying a person has subscribed or not to the marketing campaign.

1.4. Output to 'Data Mining Problem'

The trained classification model can predict an individual is subscribed to a marketing campaign or not by 'subscribed = 1' and 'non-subscribed = 0'. These models are also trained to understand the likelihood of a subscription been made by a person on future campaigns based on attributes that represent a person's demographic.

2) Data Preprocessing

2.1. Data Cleaning

The following illustrates the data cleaning process made on both Marketing and Unknown Datasets, ensuring an efficient and reliable process in constructing the training and predictive model.

Check Data Types

```
ida_data_m.dtypes
ida_data_u.dtypes
```

Data types may be stored in a different format that is related to that particular attribute. For example, numeric attributes such as ‘pdays’, ‘cons.price.idx’, ‘euribor3m’, ‘em.var.rate’ and ‘nr.employed’ that were all stored as an ‘object’ (i.e. a string) which is not appropriate to the actual value itself.

Check Missing Values

```
print(ida_data_m.isnull().sum())
print(ida_data_u.isnull().sum())
```

Missing values are crucial to building a reliable and efficient training model, preventing a slow downtime.

Check if binary values of the ‘subscribed’ attribute

```
print(ida_data_m['subscribed'].value_counts())
print(ida_data_m['subscribed'].unique())
print(ida_data_m['subscribed'].dtype)
```

In addition to checking data types, it also crucial to verify that our target variable ‘subscribed’ has been stored correctly in binary values of 0s and 1s. This ensures that the model can be trained properly due to most machine learning algorithms require only numerical input for both features and target variables.

Convert numeric attributes to its appropriate format

```
num_cols_m = ['pdays', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
for col in num_cols_m:
    ida_data_m[col] = pd.to_numeric(ida_data_m[col], errors='coerce')
    ida_data_u[col] = pd.to_numeric(ida_data_u[col], errors='coerce')
```

As previously mentioned, some machine learning algorithms require only numeric input, it is also important to convert them into their respective format, ensuring a smooth process in training the model.

Replace any missing values with NaN

```
ida_data_m.replace(["unknown", "?"], np.nan, inplace=True)
ida_data_u.replace(["unknown", "?"], np.nan, inplace=True)
```

Lastly, as we check for any missing values present in the datasets, it is ideal to replace them with a placeholder (i.e. NaN) that can be compatible in machine learning algorithms, enabling free of errors and a smooth model training.

2.2. Normalisation

This technique enables a consistent scale to utilise the StandardScaler, which performs feature scaling for numeric attributes. This process is significant for classification models, such as, K-Nearest Neighbour, Support Vector Machines and Neural Networks that are sensitive to vast amount of values, causing skewness in results.

Normalisation was applied after building the preprocessing pipeline for numerical columns demonstrated below:

For KNN, SVM and NN:

```
num_transformer = Pipeline(steps=[
    (                                                    ),
    ('scaler', StandardScaler())
])
```

The ‘num_transformer’ variable sets up a pipeline by scaling the numerical features using StandardScaler, ensuring consistency during model training for KNN, SVM and NN.

For DT and RF:

```
num_transformer_tree = Pipeline(steps=[
    (                                                    ),
    ('scaler', StandardScaler())
])
```

The ‘num_transformer_tree’ variable uses the same logic sets up a pipeline to the other classifiers, such as KNN, SVM and NN, however for decision trees and random forest. This is to ensure consistency throughout during model training g for DT and RF.

2.3. Data Transformation

Following from normalising all classifiers, data transformation starts from pipelining numerical and categorical features, then initiating partitioning process and fitting and transforming preprocessors for all classifiers demonstrated below:

Step 1: Pipelining

Classifiers: K-Nearest Neighbour, Support Vector Machine & Neural Network

- **Numerical Features**

```
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

Using 'X_current' column list, imputation was performed by replacing missing values of numerical columns to the column average using SimpleImputer. This means the missing data will be replaced with the mean of the specific column. Normalisation (i.e. scaling) categorical features also performed using StandardScaler. Overall, this ensures all missing values are being handled for smoother model training.

- **Categorical Features:**

```
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

Using the 'X_tree' column list, imputation was performed by replacing missing values of categorical columns to 'most_frequent" using SimpleImputer. This means missing data will be replaced with value that mostly frequent in that specific column. Instead of normalisation, categorical features are required to be converted into binary format, thus, One Hot encoding was performed using OneHotEncoder, eliminating ordinality that may be misinterpreted by the model enabling a much more efficient process overall.

- **Combining Preprocessors:**

```
preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_cols),
    ('cat', cat_transformer, cat_cols)
])
```

This step combines both pipelining processes performed for numerical and categorical features and is applied to the appropriate columns of the dataset. This preprocessor is then used for fitting and transforming onto the training data.

Classifiers: Decision Tree and Random Forest

Pipelining is not required for Decision Trees and Random Forest, however is logical to perform this step similarly to KNN, SVM and Neural Network, providing consistent scaling and ensures any missing data are handled well that can mitigate errors during model training. Instead it is labelled as 'num_transformer_tree' and 'cat_transformer_tree' to distinguish from transforming data with KNN, SVM and NN.

- **Numerical Features:**

```
num_transformer_tree = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

Like pipelining KNN, SVM and NN, perform imputation and normalisation for classification models, DT and RF using SimpleImputer and StandardScaler respectively.

- **Categorical Features:**

```
cat_transformer_tree = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

Like pipelining KNN, SVM and NN, perform imputation and One Hot encoding for classification models, DT and RF using SimpleImputer and OneHotEncoder respectively.

- **Combining Preprocessors:**

```
preprocessor_tree = ColumnTransformer(transformers=[
    ('num', num_transformer_tree, num_cols_trees),
    ('cat', cat_transformer_tree, cat_cols_trees)
])
```

Like combining pipelining processes for numerical and categorical features, a similar process is done for DT and RF.

<u>Step 2: Partitioning for Validation Testing</u>

Classifiers: K-Nearest Neighbour, Support Vector Machine & Neural Network

```
X_train, X_val, y_train, y_val = train_test_split(
    X_current, y, test_size=0.3, stratify=y, random_state=42)
```

Classifiers: Decision Trees and Random Forest

```
X_train_tree, X_val_tree, y_train_tree, y_val_tree = train_test_split(
    X_tree, y, test_size=0.3, stratify=y, random_state=42)
```

A 70/30 train-test split was performed where 70% of the data in marketing dataset were retained for training, whereas 30% of the data were used for validation testing, assessing the model's performance. The purpose of training 70% of the data instead of 50/50 split due to mitigating the risk of underfitting data and high bias, such that the model will not have enough patterns to learn from. This is also vice versa to having too much training data as this causes overfitting. Also, this leaves too little for validation, to be able to judge model performance for parameter tuning and feature selection. Overall, this current split helps to build a model that has adequate examples of learning robust patterns and provides an unseen data to better accurately predict real-world problems.

Step 3: Fitting and Transforming Data

Classifiers: K-Nearest Neighbour, Support Vector Machine & Neural Network

```
X_train_processed = preprocessor.fit_transform(X_train)
X_val_processed = preprocessor.transform(X_val)
```

Classifiers: Decision Trees and Random Forest

```
X_train_tree_processed = preprocessor_tree.fit_transform(X_train_tree)
X_val_tree_processed = preprocessor_tree.transform(X_val_tree)
```

This last step applies the full preprocessing steps performed in the above into a training dataset. The 'fit_transform' method is executed through learning the parameters from the training set only, whereas the 'transform' method allows the learned parameters to convert the validation data. This transformation process is a crucial step as this avoids data leakage, meaning the future unseen data cannot influence how the model is trained.

2.4. Linear Correlation between numeric and 'subscribed' attributes

- Discuss linear correlation was performed and how its used for building classifiers, particularly in the Section 4
- Include correlation matrix

Linear Correlation analysis is significant at this stage of the model development process, due to determining feature selection. This allows to observe the correlation between two variables, particularly an attribute with the 'subscribed' attribute to deduce whether these attributes can be trained in the model to accurately predict the number of subscribers, as well as labelled a subscriber of the marketing campaign. This analysis can also assist in identifying classification models, such as KNN, SVM and NN that struggle with multicollinearity.

This stepwise approach of plotting a linear correlation matrix and the analysis is shown below:

Step 1: Merge ‘subscribed’ column again into a separate Dataframe for analysis

```
# Step 1: Merge X and y back into a single Dataframe
corr_df = X.copy()
corr_df['subscribed'] = y
```

Step 2: Ensure numeric columns in appropriate format

```
# Step 2: Ensure numeric columns in appropriate format
num_corr_df = corr_df.select_dtypes(include=['int64', 'float64'])
```

Step 3: Plot Linear Correlation Matrix

```
# Step 3: Linear correlation representing numeric variables and 'subscribed'
correlation_matrix = num_corr_df.corr(method='pearson')

plt.figure(figsize=(12,10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0, square=True, linewidths=0.5)
plt.title("Correlation Matrix between numeric variables and 'subscribed'", fontsize=14, fontweight='bold')
plt.tick_params(axis='both', which='both', length=0)
plt.tight_layout()
plt.show()
```

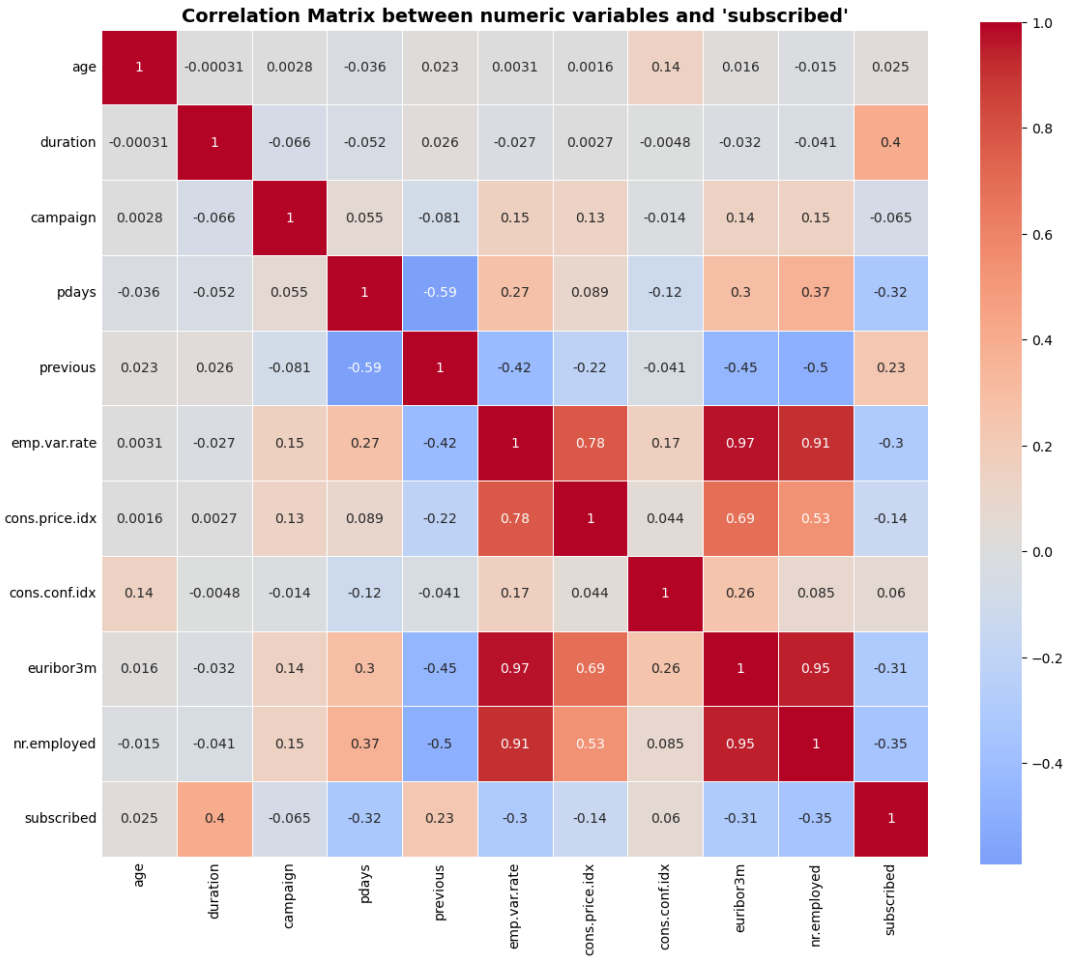

Figure 1. Correlation matrix between numeric and ‘subscribed’ attributes

The correlation matrix in Figure 1 highlights the distinctive correlation values and linearly relationships between attributes, guiding in the process of feature selection and handling multicollinearity. Most attributes were kept to effectively maintain model development and training, mitigating the risk of underfitting that may increase bias, as well as overfitting models. The attributes that exhibited a positive linear relationship with ‘subscribed’ are ‘duration’ and ‘previous’ and attributes that exhibited a negative linear relationship are ‘pdays’, ‘emp.var.rate’, ‘cons.price.idx’, ‘euribor3m’ and ‘nr.employed’. The attributes dropped from the training data were attributes that displayed a negative linear relationship to ‘subscribed’, these include ‘emp.var.rate’, ‘euribor3m’, age’ and ‘cons.conf.idx’. When analysing the linear relationships between certain attributes this portrays a higher risk for multicollinearity. These attributes are between ‘euribor3m’ and ‘emp.var.rate’ together, the correlation value shows 0.97, between ‘euribor3m’ and ‘nr.employed’ shows 0.95 and lastly between ‘emp.var.rate’ and ‘nr.employed’ shows 0.91. Hence, the attributes that were dropped from the training data were ‘emp.var.rate’ and ‘euribor3m’ as the correlation value between these with ‘subscribed’ indicate that there will be less accurate predictions compared to ‘nr.employed’. The attribute ‘age’ was dropped as well, due to being a continuous variable which will not work well with classifiers; KNN and DT however most of the dataset contain several continuous attributes which is difficult to drop due to underfitting.

3) Problem-Solving Approach – Part 1: Data Mining Preparation

3.1 Feature Selection

Several continuous attributes in the dataset that will not cooperate well with classifiers; KNN and DT. However, it is impossible to drop due to underfitting the other classifiers, especially complex models like SVM and NN. Based on Figure 1 (correlation matrix), attributes that were dropped either displayed weaker negative correlations or the correlation value was close to zero with the 'subscribed' attribute that are unable predict whether an individual has been subscribed or not. Attributes, such as 'row ID', 'emp.var.rate', 'euribor3m', 'age', 'cons.conf.idx were dropped for KNN, SVM and NN. On the contrary to DT and RF, the only attributes that are dropped are 'row ID', 'age', 'cons.conf.idx' as they handle multicollinearity well, whilst the training the model. This aids to reduce redundancy and increase the effectiveness of classifiers to build and utilise to predict accurate 'subscribed' attribute values.

3.2 Parameter Optimisation

Table 1. Summary of Selected Parameter Settings for each Classifier

Classifier	Parameters	Parameter Settings tested	Best Settings Selected	Justification
K-Nearest Neighbour	*n_neighbours*	k_values = [3, 5, 7, 9]	k=5	Focused with odd values that were also not too large to train with, using the imbalanced dataset. The purpose of odd values used is to avoid ties and reduce underfitting. However, very small values were not used to avoid overfitting and are sensitive to noisy data. The optimal k value was 5 that resulted with the highest F1 score of 0.4912.
Decision Tree	*max_depth*	depths = [3, 5, 7, 10, 15]	depth = 3	Tested with a range of values between 3 (shallow) and 15 (deep) to understand any chances of overfitting/underfitting as the tree grows. As the tree further grows for imbalanced data, the higher risk of overfitting. When testing out these settings for imbalanced and balanced data, the results displayed that using imbalanced data gave the highest F1-score of 0.6129.
Random Forest	*n_estimators, max_depth*	n_estimators = [50, 100, 150] max_depths = [5, 10, 15]	n_estimators = 150 max_depth = 15	Tuned two hyperparameters that would influence the model's performance, such as the number of decision trees within a forest (n_estimators) and the depth of tree can grow (max_depths). As a tree becomes more complex, this risks overfitting for imbalanced data.

				However, the balanced data was capable of a complex tree model. This emphasises the model is capable of predicting accurately and simultaneously learning from minority classes. Hence, this produced an F1-score of 0.6212.
Support Vector Machine	*kernels, C*	kernels = ['linear', 'rbf'] C_values = [0.1, 1, 10]	Kernels = rbf C = 1	Tuned two types of kernels; linear and RBF on the balanced data. Based on the highest F1-score of 0.5770, the results had shown that RBF kernels were the best settings to capture more complex, non-linear boundaries. The results also shown a C value of 1 is better as this can accurately predict the 'subscribed' attribute values.
Neural Network	layer_sizes, *alphas*	layer_sizes = [(50,), (100,), (100, 50)] alphas = [0.0001, 0.001]	hidden_layer_sizes = (50,) Alpha = 0.0001	The model has displayed the highest F1-score of 0.5863 on the balanced dataset using SMOTE. Simply, this justifies that simple architecture with one hidden layer of 50 units performed best as this reduced the risk of overfitting on fewe minority classes. The balancing technique, SMOTE, has shown slight improvement from imbalanced dataset and balanced dataset with RandomOverSampler.

3.3 Handling Unbalanced Data

Following from tuning hyperparameters for the 5 classifiers, as well as finding the optimal settings by balancing data on certain classifiers, it had evidently shown small margin of improvement. There were 3 classifiers that had used balanced datasets to obtain optimal settings, such as RF, SVM and NN, however NN had particularly used SMOTE to perform balancing.

4) Problem-Solving Approach – Part 2: Classification Implementation

4.1 K-Nearest Neighbour (KNN)

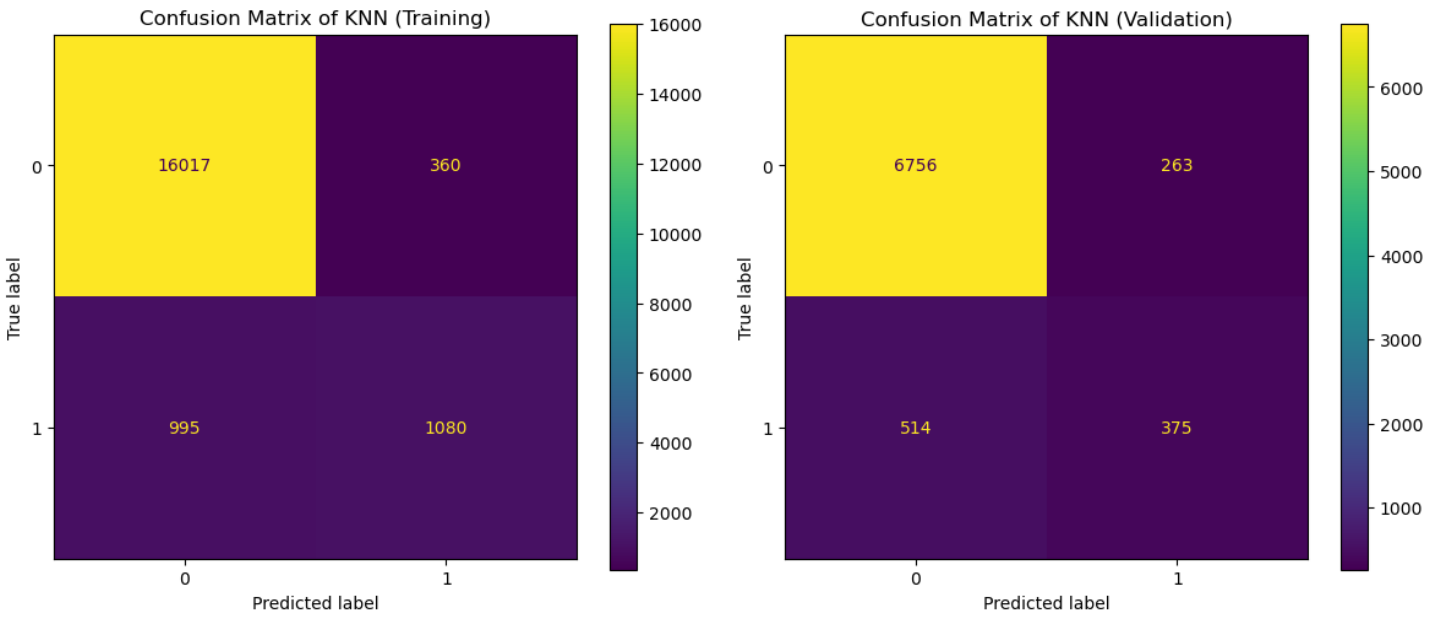

Figure 2. Confusion Matrix of KNN (Training & Validation)

This model was able to predict most of the non-subscribers where the total number of predictions made is 16017 and 6756 for training and validation respectively. However, the validation set struggles to predict subscribers more than the training set, as the total number of subscribers counted was only 375, compared to the count in training set at 1080. This illustrates that KNN performs well at predicting the majority class (non-subscribers) compared to the minority class (subscribers) typical for this imbalanced dataset, explaining the moderate F1-score of 0.4912.

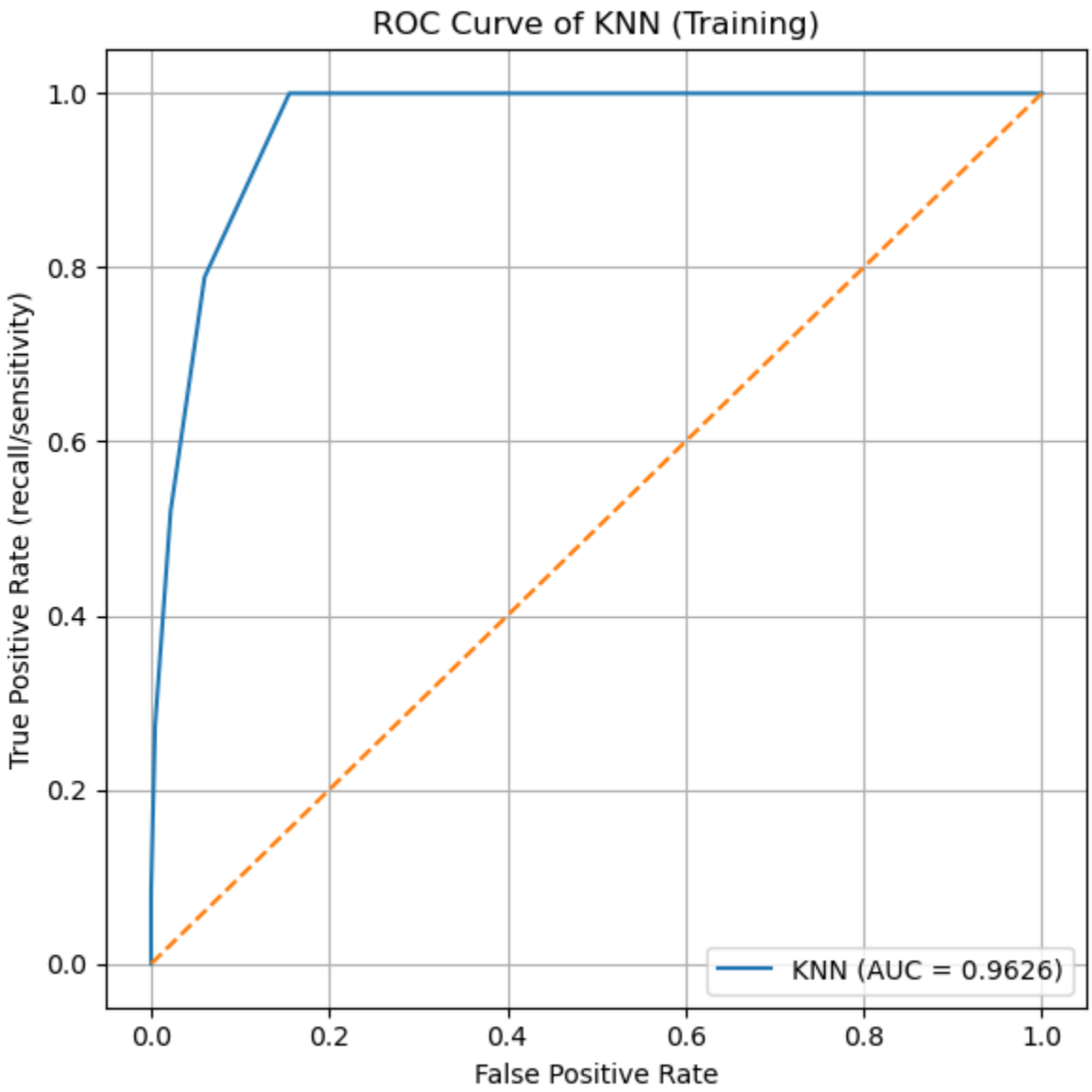

Figure 3. ROC graph of KNN (Training)

This ROC graph above illustrates the training set has a better attempt at accurately predicting between subscribers and non-subscribers, as this resulted in the AUC value of 0.96. It is also evident on the blue curve is much further away from the orange diagonal line.

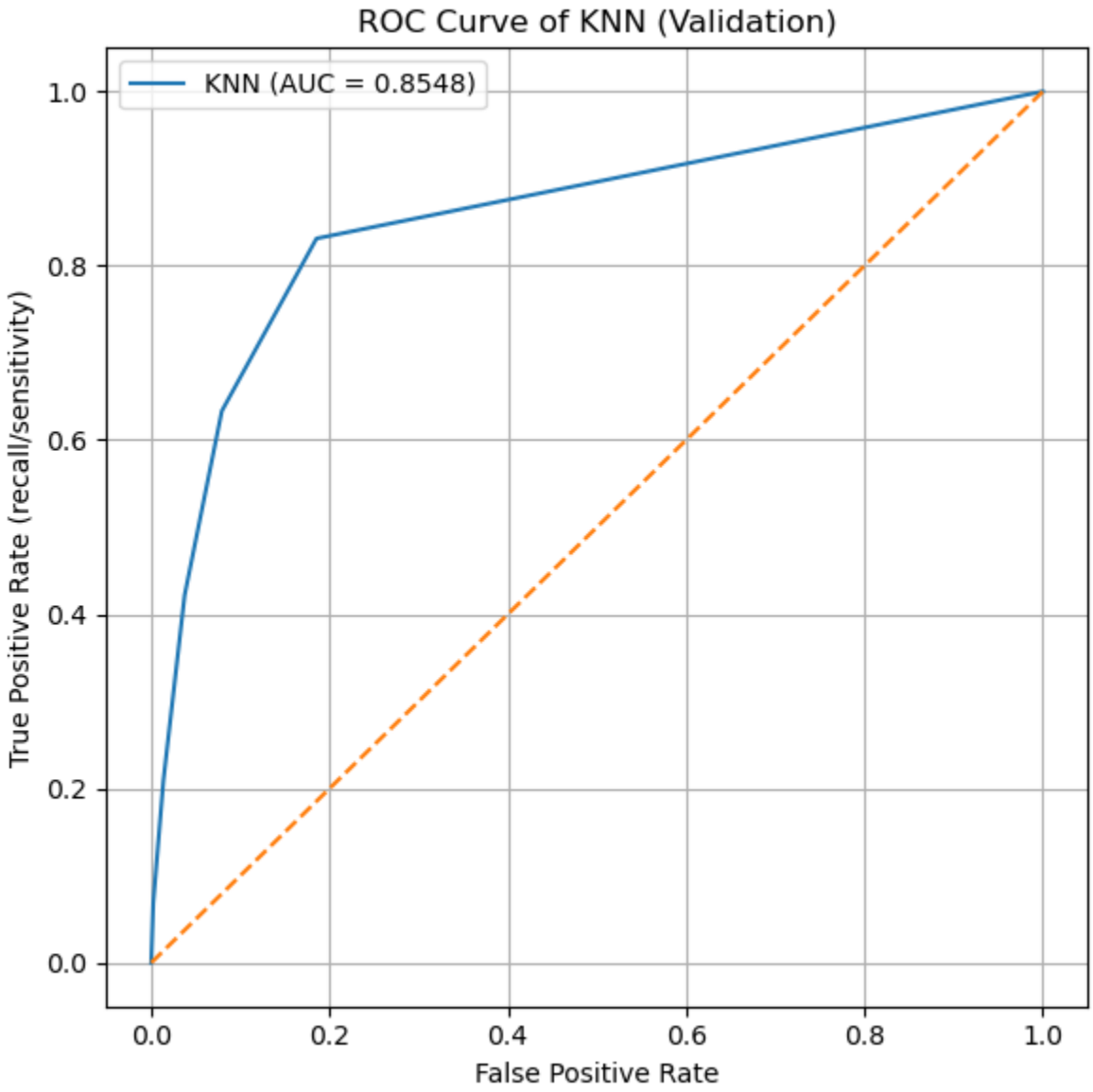

Figure 4. ROC graph of KNN (Validation)

In comparison to Figure 3, this graph shows the blue curve line is closer to the orange diagonal line. This exhibits that this model struggles with classifying between subscribers and non-subscribers. In addition to this fact, this can be assumed that there are more False Negative values, as a result an AUC value of 0.85 is shown here which more lower than the training set.

Table 2. Summary of Training & Validation Results

	Recall/Sensitivity	Precision	F1-Score
Training (non-subscribed)	0.98	0.94	0.96
Validation (non-subscribed)	0.93	0.96	0.95
Training (subscribed)	0.52	0.75	0.61
Validation (subscribed)	0.42	0.59	0.49

Table 3. Summary of Accuracy and AUC values for Training & Validation

	Accuracy	AUC
Training	0.93	0.96
Validation	0.90	0.85

In conclusion, this classifier may not be ideal for using to make predictions on whether a person is subscribed or not to the marketing campaign. This can be seen where the recall value is 0.52 compared to the training set at 0.98, indicating it cannot correctly identify all actual subscribers, as well as the precision of this model is 20% difference in making accurate predictions.

4.2 Decision Trees

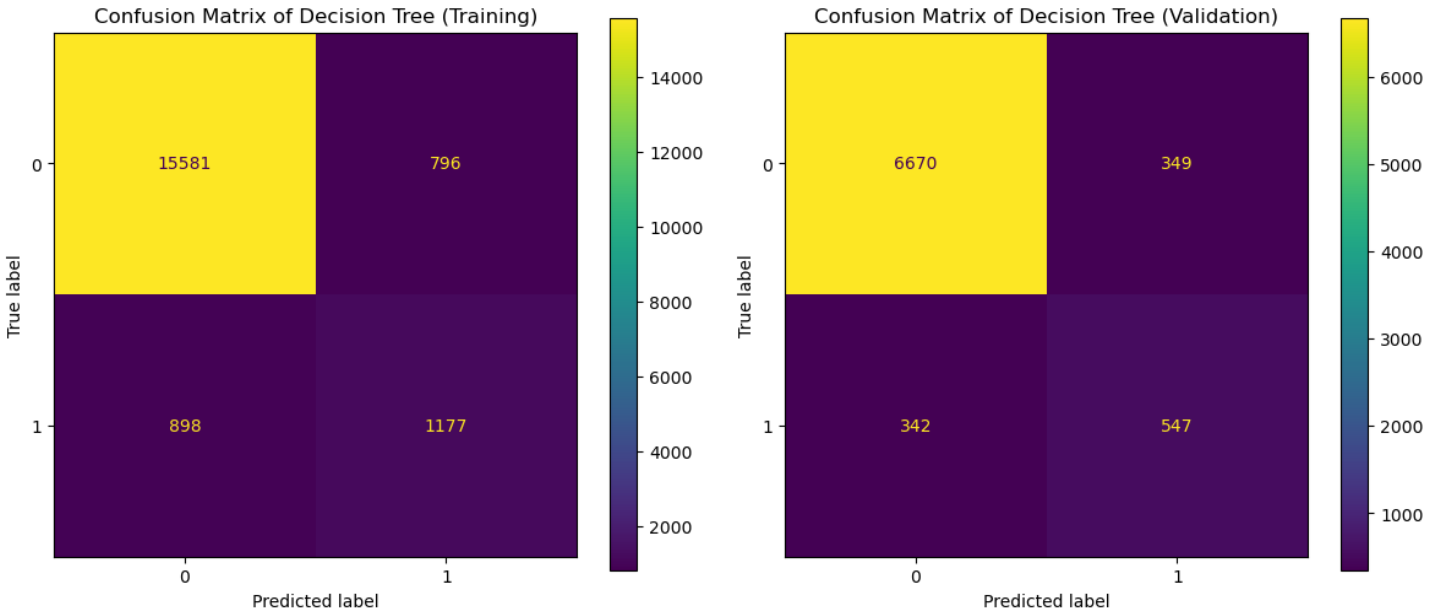

Figure 5. Confusion Matrix of Decision Trees (Training & Validation)

This model suggests that 547 subscribers are predicted from the validation set, whereas 1177 subscribers are predicted from the training set, meaning this model is quite ideal for make accurate predictions between the number of subscribers to non-subscribers. This indicates that there is a low False Positive rate, that balances with the True Positive rate, however some subscribers are still missed as it can be seen a False Positive value of 349 in the validation set.

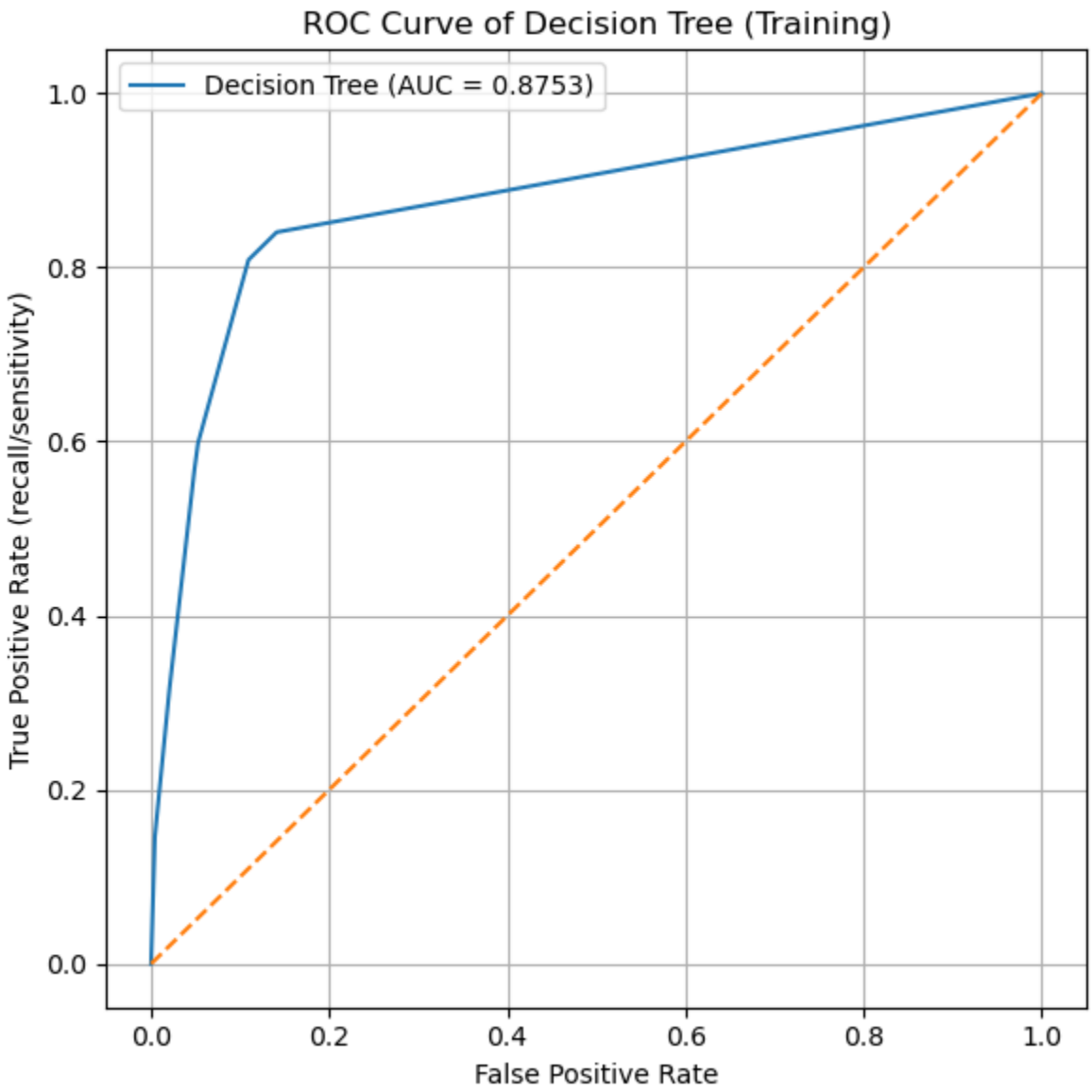

Figure 5. ROC graph of Decision Trees (Training)

This ROC graph above illustrates that the training set has a good attempt at identifying subscribers to non-subscribers where the blue curve line is farther away from the orange line. It has a moderate True Positive rate as it can be seen here. The AUC value is shown at 0.87 which is still has a decent performance in make accurate predictions for the minority class (non-subscribers).

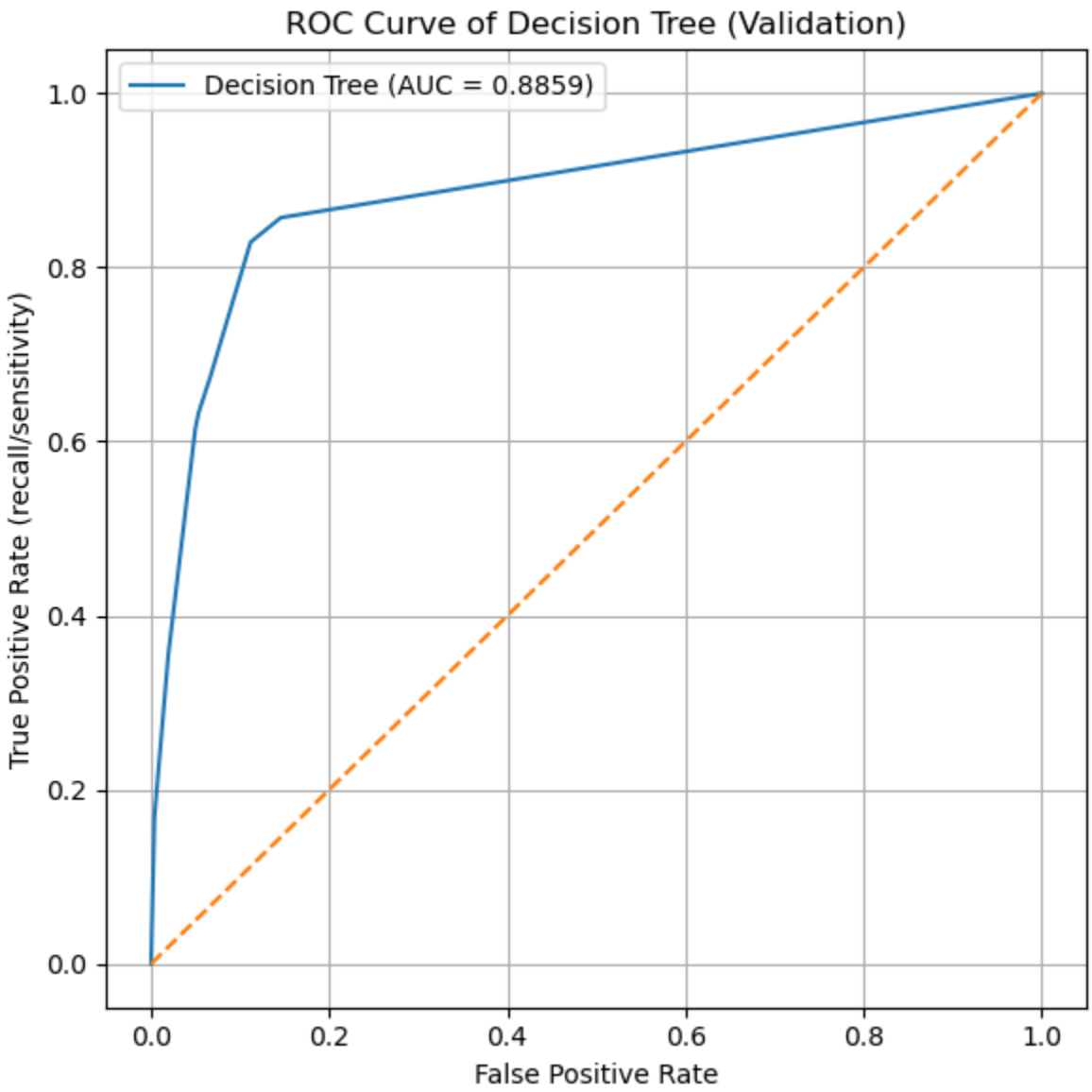

Figure 6. ROC graph of Decision Trees (Validation)

In comparison to Figure 5, the validation results show a higher AUC value of 0.88 which shows slightly better performance compared to the training set. It is also visible with the blue curve bending farther away from the orange line more than the training set. This indicates that the model can predict more accurately on whether a person is a subscriber or not, making this ideal classifier for future predictions.

Table 4. Summary of Training & Validation Results

	Recall/Sensitivity	Precision	F1-Score
Training (non-subscribed)	0.95	0.95	0.95
Validation (non-subscribed)	0.95	0.95	0.95
Training (subscribed)	0.60	0.57	0.58
Validation (subscribed)	0.61	0.62	0.61

Table 5. Summary of Accuracy and AUC values for Training & Validation

	Accuracy	AUC
Training	0.91	0.88
Validation	0.91	0.89

Although the AUC value in the validation than the training set, the precision and recall is still much lower. This can indicate that there's overfitting in the training dataset. The F1-score shows higher in the training set, however decreased in validation, as this can mean the model is ranked positive only at a specific threshold, meaning inconsistency throughout all thresholds. The validation set might also have less ambiguous examples that make the AUC appear to be better, as it can confidently distinguish between classes. Hence, this classifier may be ideal on the surface but only for unseen data.

4.3 Random Forest

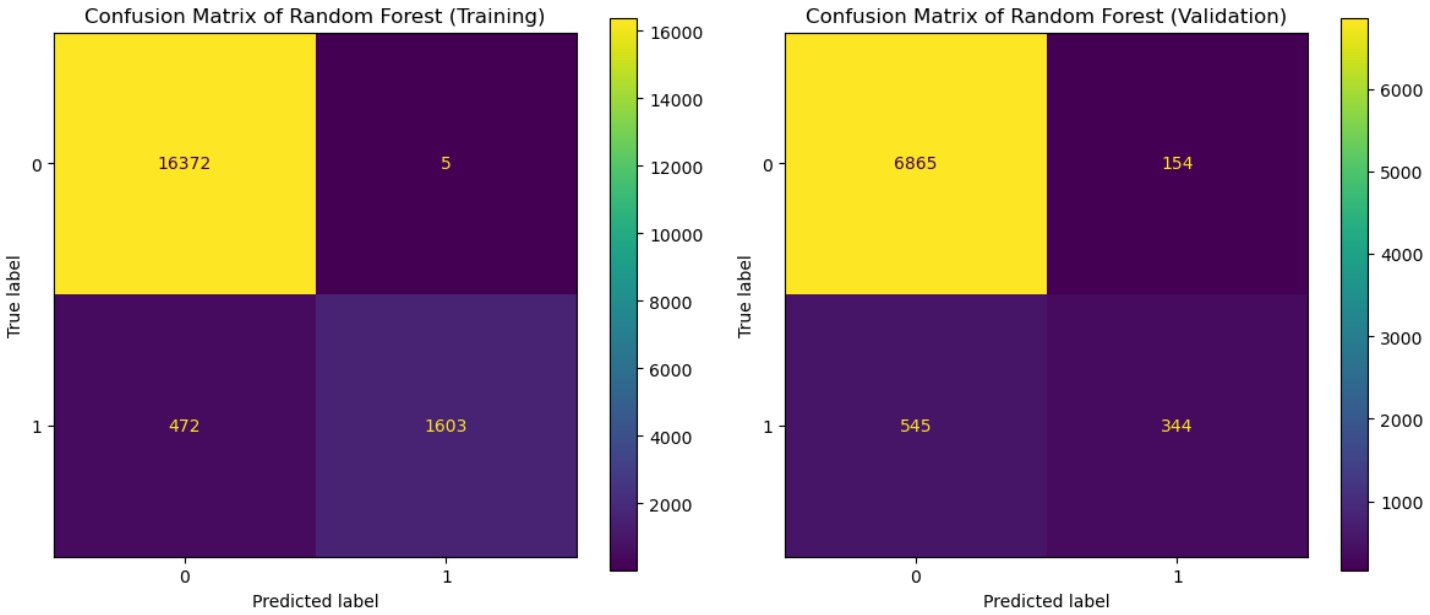

Figure 7. Confusion Matrix of Random Forest (Training & Validation)

The training set predicts more accurately by distinguishing between subscribers to non-subscribers properly, where counted 16372 non-subscribers and 1603 subscribers. However, the validation set can only predict non-subscribers accurately but makes poorer predictions on subscribers and there is a higher True Negative rate.

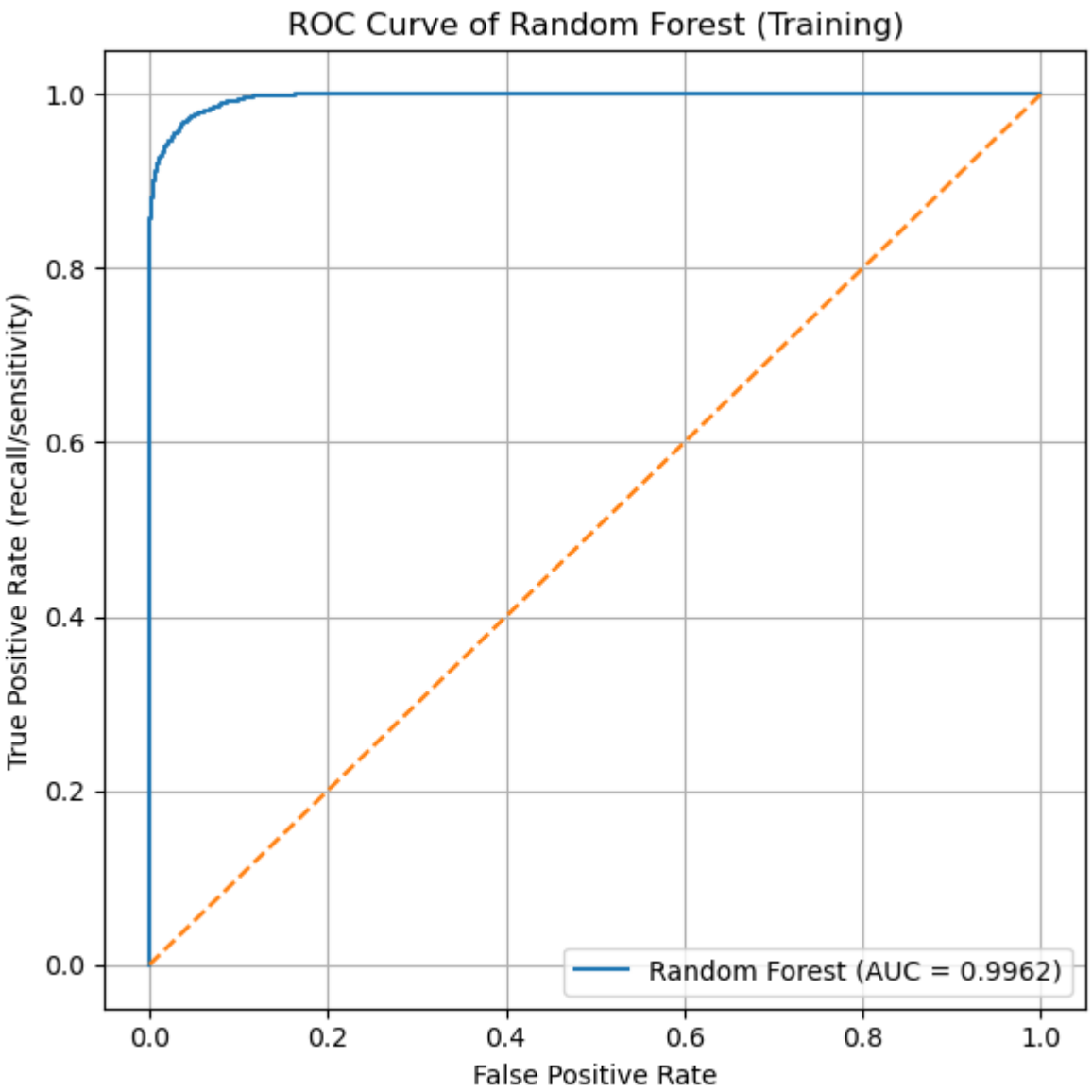

Figure 8. ROC graph of Random Forest (Training)

The ROC graph presents a smooth curve, showing a very good True Positive rate as evident in the AUC value of 0.99. In addition, this illustrates that the model can accurately distinguish between subscribers to non-subscribers.

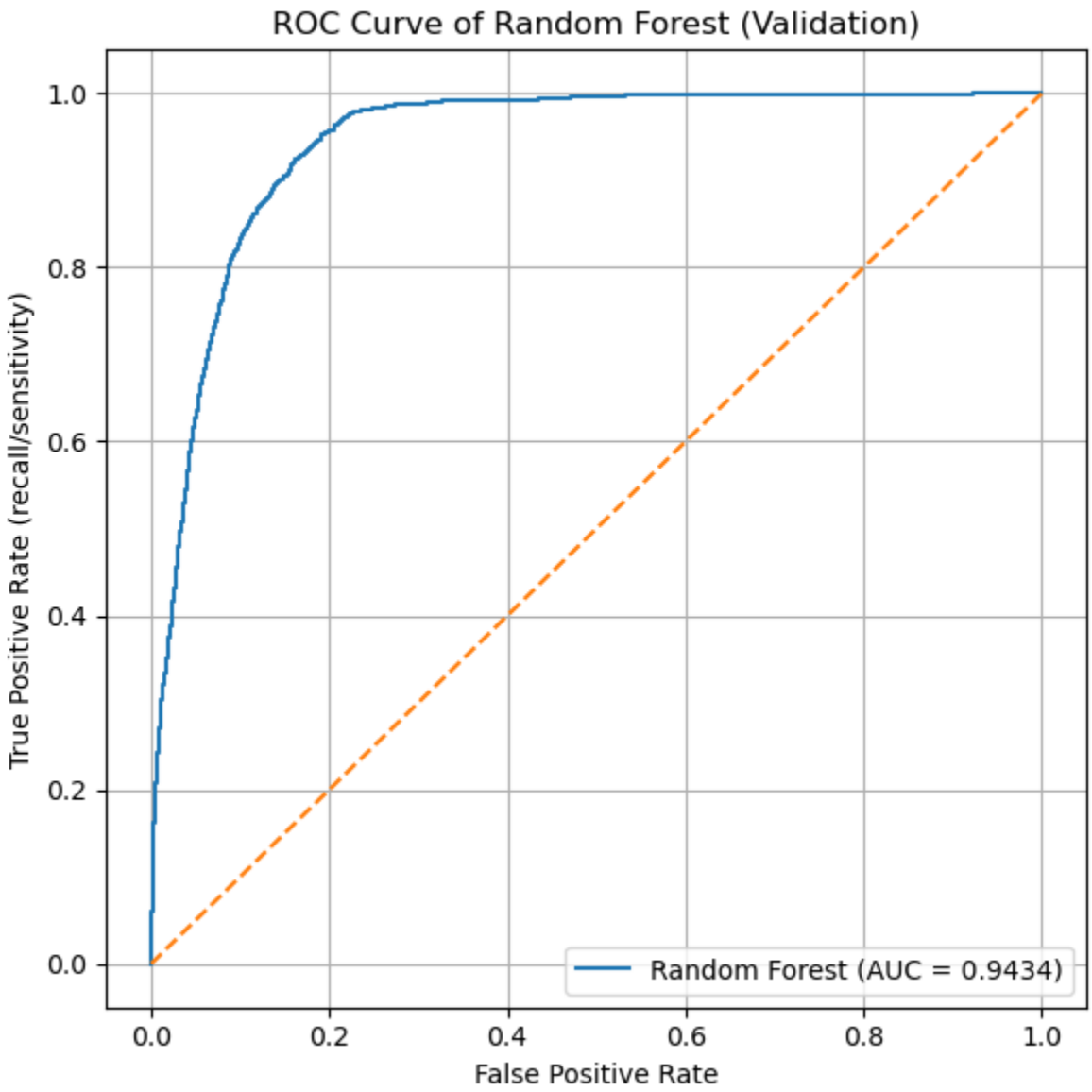

Figure 9. ROC graph of Random Forest (Validation)

In comparison to Figure 8, the validation set can also predict and distinguish between subscribers and non-subscribers almost perfectly, with an AUC of 0.94.

Table 6. Summary of Training & Validation Results

	Recall/Sensitivity	Precision	F1-Score
Training (non-subscribed)	1.00	0.97	0.99
Validation (non-subscribed)	0.93	0.98	0.95
Training (subscribed)	0.77	1.00	0.99
Validation (subscribed)	0.69	0.39	0.50

Table 7. Summary of Accuracy and AUC values for Training & Validation

	Accuracy	AUC
Training	0.97	0.99
Validation	0.91	0.94

On the contrary with the accuracy and AUC values for both sets are high and above 0.90, it indicates the model's performance of accurately predicting subscribers still needs improvement, as the precision is at 0.39. This can potentially mean that there could be an underfitting or overfitting of the model.

4.4 Support Vector Machine (SVM)

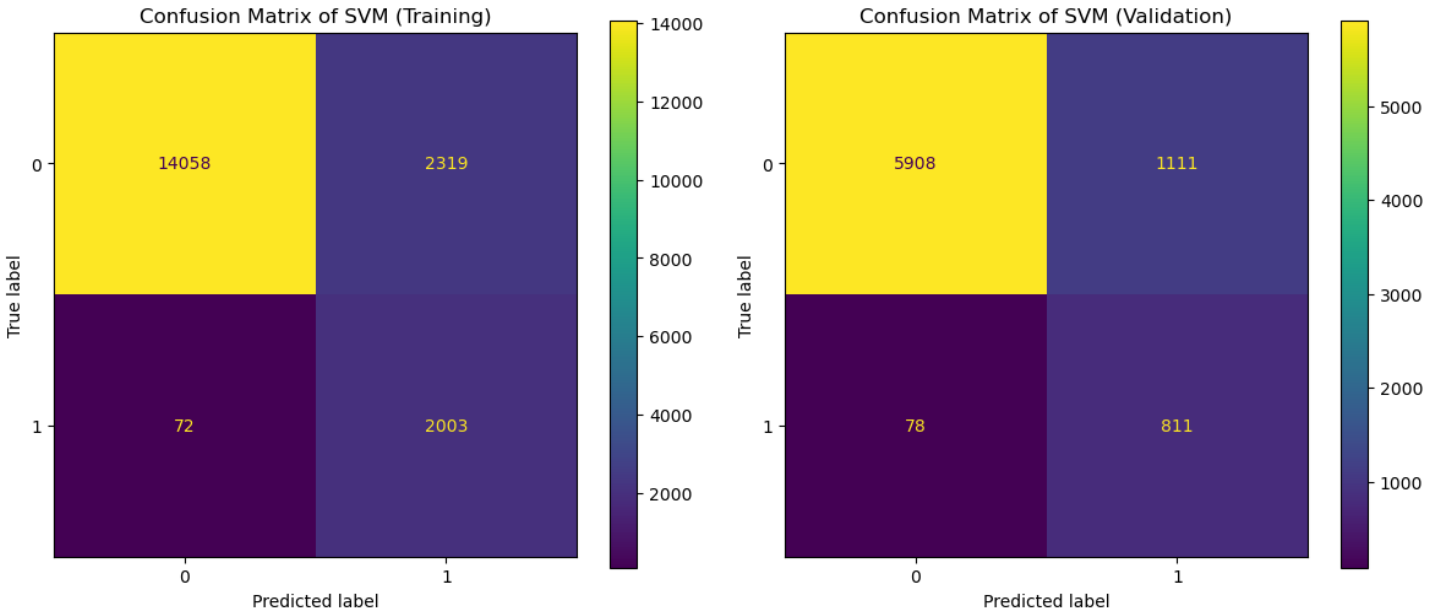

Figure 10. Confusion Matrix of SVM (Training & Validation)

This model presents that a higher number of counts for both subscribers and non-subscribers, meaning it can accurately distinguish between subscribers to non-subscribers. However, there's also a higher False Positive rate, highlighting that the model is more biased towards more subscribers than not, leaving with very few non-subscribers.

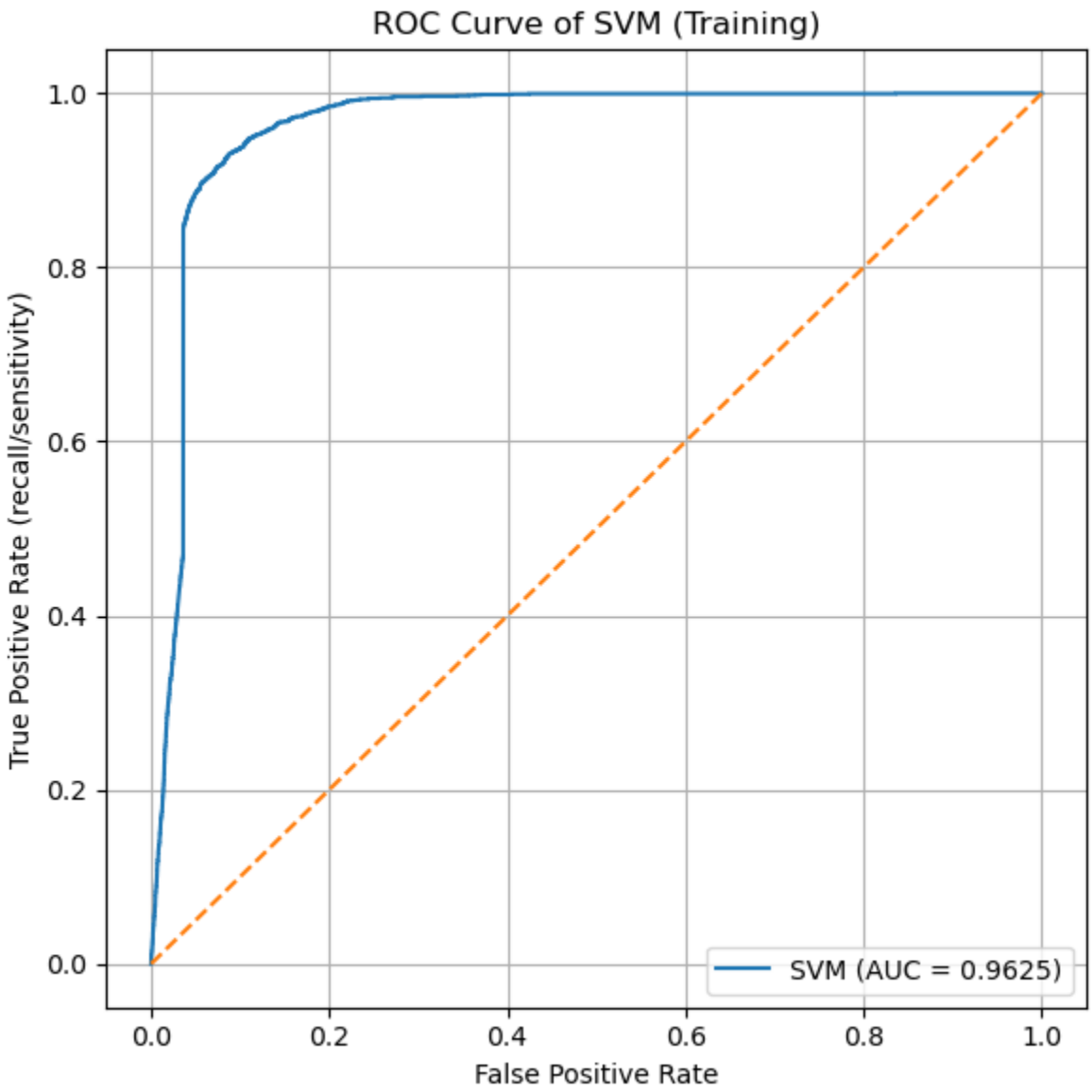

Figure 11. ROC graph of SVM (Training)

The ROC graph represents the blue curve being further away from the orange line meaning there's mostly accurate predictions between subscribers to non-subscribers. However, the graph also exhibits a slight shift towards the False Positive axis, which matches the idea that the model can predict more subscribers than not.

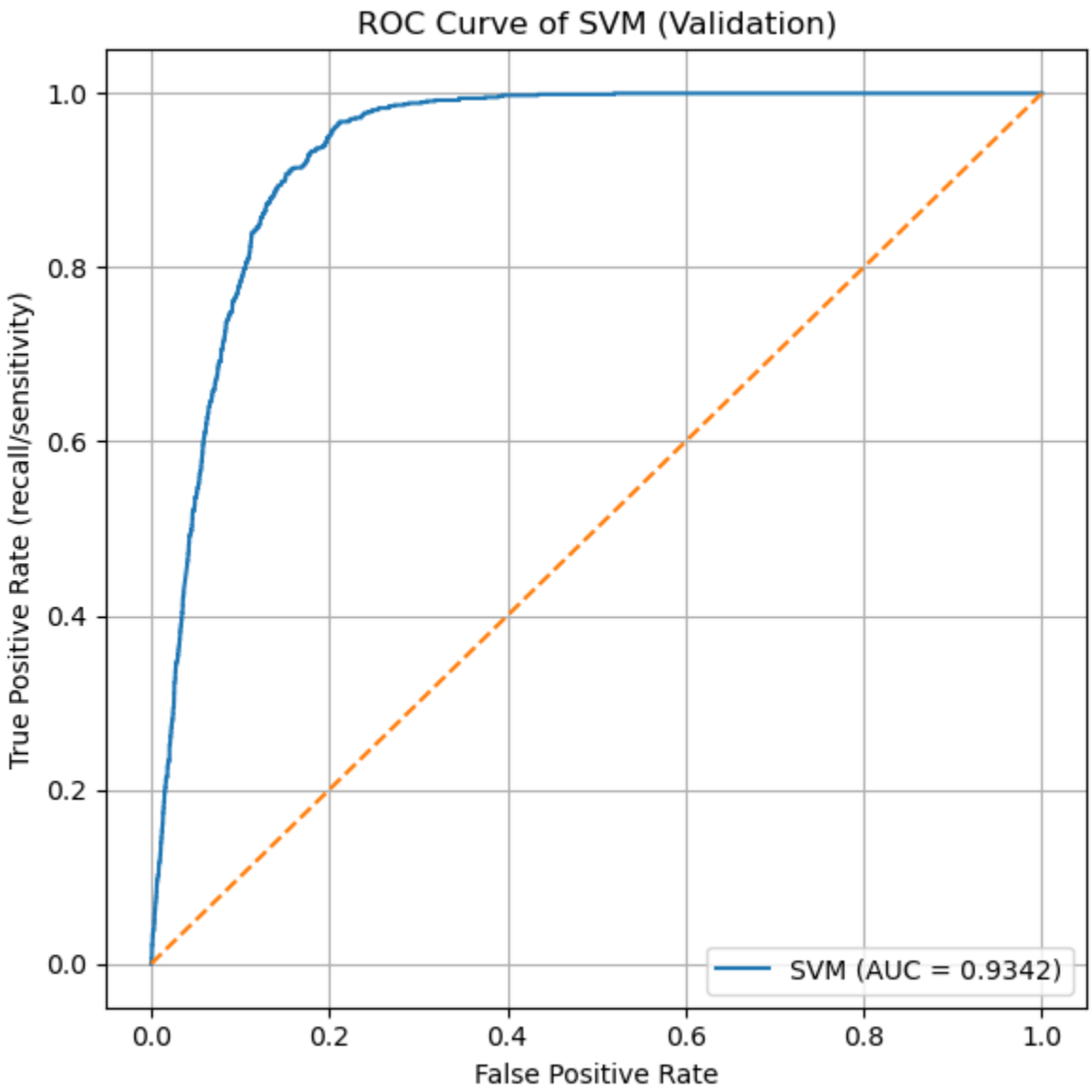

Figure 12. ROC graph of SVM (Validation)

In comparison to Figure 11, the graph shows that more false positives are made compared to true positives, evident in the AUC value of 0.93, emphasising that more subscribers are predicted inaccurately compared to non-subscribers. This highlights may need further tuning on parameter settings to find the best approach to lean towards more non-subscribers.

Table 8. Summary of Training & Validation Results

	Recall/Sensitivity	Precision	F1-Score
Training (non-subscribed)	0.86	0.99	0.92
Validation (non-subscribed)	0.99	0.84	0.91
Training (subscribed)	0.46	0.97	0.63
Validation (subscribed)	0.42	0.91	0.58

Table 9. Summary of Accuracy and AUC values for Training & Validation

	Accuracy	AUC
Training	0.87	0.96
Validation	0.85	0.93

The recall value indicates similarly to the Figure 11 and 12, in that there are more false positives being predicted compared to true positives, as non-subscribers were very few. As the recall is lower, the precision is much higher as the total counts of all positive cases is higher compared to negative.

4.5 Neural Network

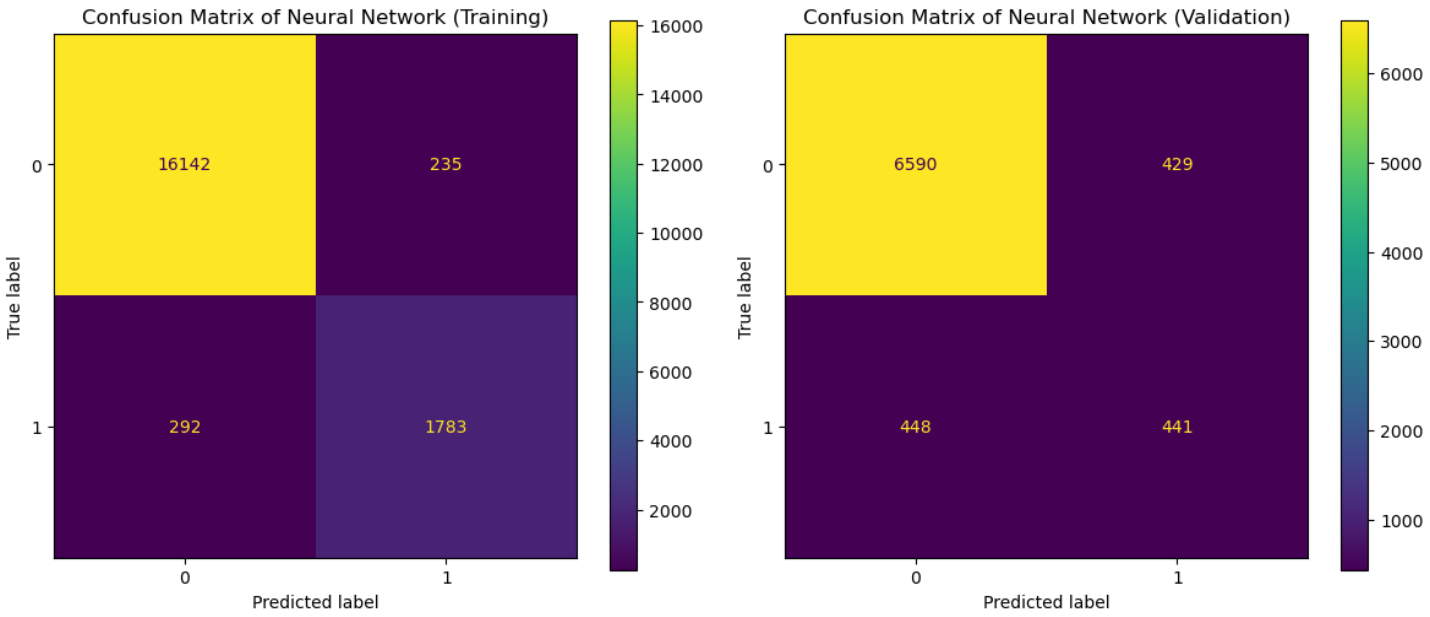

Figure 13. Confusion Matrix of Neural Network (Training & Validation)

This presents a lower number of true positive cases made for validation compared to training set. However, the number of non-subscribers is still accurately predicted properly. This highlights that the model cannot predict correctly the number of actual subscribers.

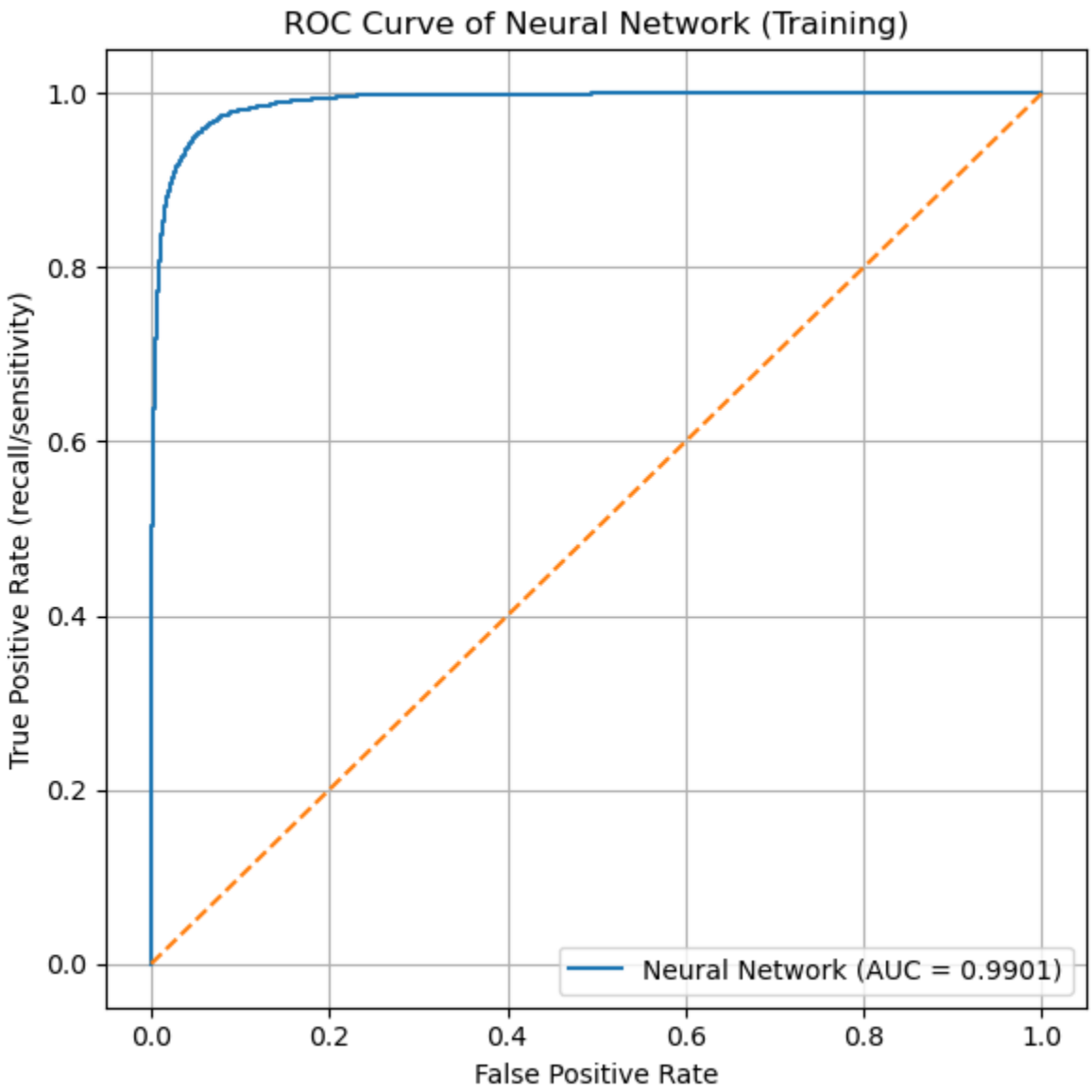

Figure 14. ROC graph of Neural Network (Training)

This graph represents a smooth curve of true positive cases made in the training set, as this indicates the model can accurately predict the number of non-subscribers to subscribers, where this is represented by the AUC value of 0.99.

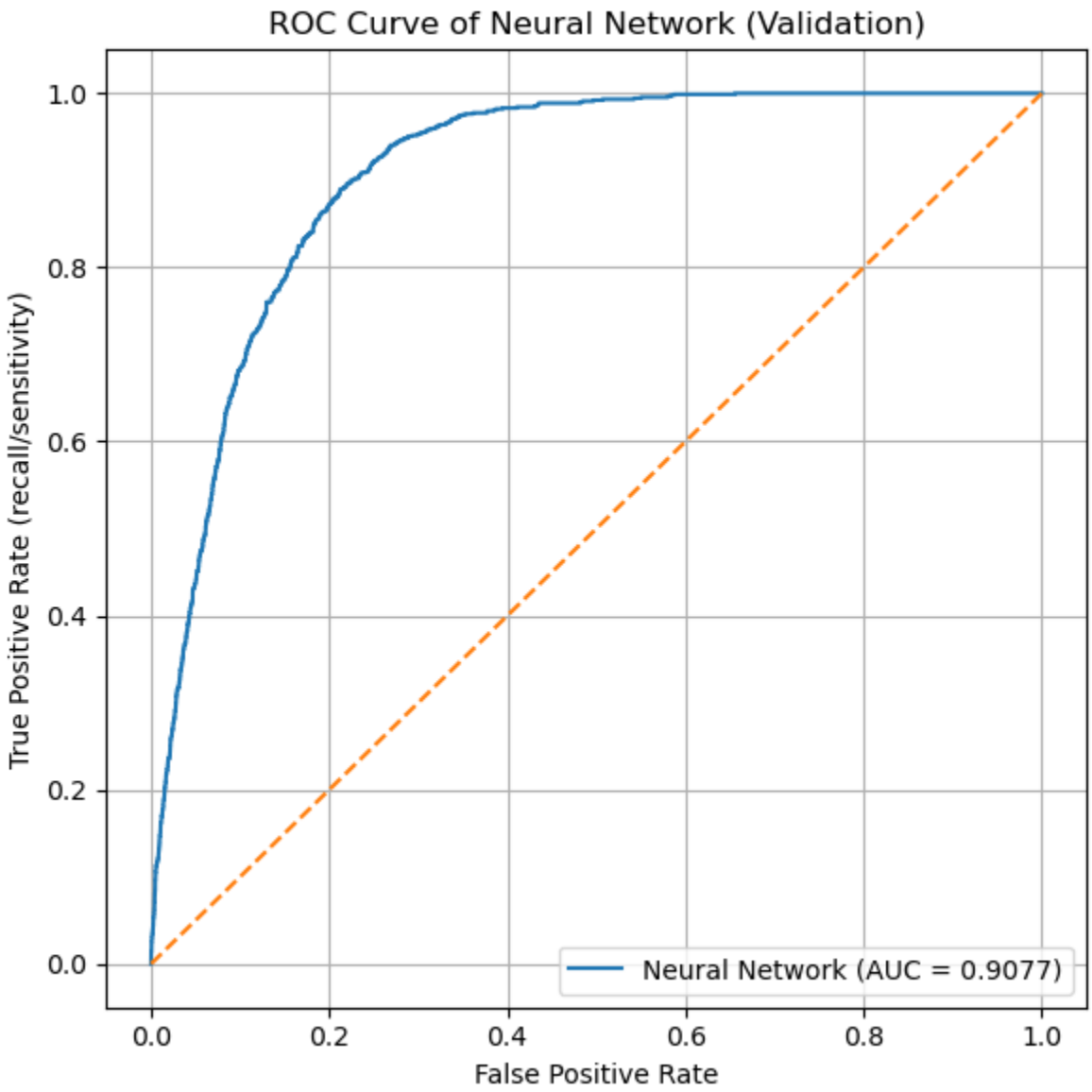

Figure 15. ROC graph of Neural Network (Validation)

In comparison to Figure 14, the blue curve shows more false positive cases are being predicted, which is logical from the graph representing the curve closer to the orange line. This explains with the lower AUC value compared to Figure 14.

Table 11. Summary of Training & Validation Results

	Recall/Sensitivity	Precision	F1-Score
Training (non-subscribed)	1.00	0.97	0.99
Validation (non-subscribed)	0.93	0.98	0.95
Training (subscribed)	0.77	1.00	0.99
Validation (subscribed)	0.69	0.39	0.50

Table 12. Summary of Accuracy and AUC values for Training & Validation

	Accuracy	AUC
Training	0.97	0.99
Validation	0.91	0.94

The precision value is lower in validation (subscribed) compared to training (subscribed), however the recall value is only slightly in training (subscribed) compared to validation (subscribed). This accentuates that the model can predict all positive cases regardless being true cases or not and is also quite good at predicting only true positive cases.

4.6. Evaluation

The selected classifier that was the best at predicting between subscribers and non-subscribers after analysing the validation results is Support Vector Machine. This classier was chosen due to its strong performance on validation testing, particularly handling complex and non-linear relationships within the dataset. Comparatively to other classifiers, it did not have the best validation F1-score, however it has balanced trade-off between precision and recall, particularly for minority class for subscribers. The architecture of the Support Vector Machine had the ability to predict better with subscribers with higher positive cases, whilst still predicting similar number of counts on non-subscribers like other classifiers. Although Support Vector Machine may inaccurately predict incorrect subscribers, plus slightly intensive in computation, it provides a well-rounded and reliable performance for identifying potential subscribers, making this a preferred model for solving this data mining problem.

5) Kaggle Submission Score of all models

5.1. Best model based on Kaggle Score

Table 13. Assessment Score for all classifiers

Classifier	Score
K-Nearest Neighbour	0.45749
Decision Trees	0.57801
Random Forest	0.50331
Support Vector Machine	0.58666
Neural Network	0.53421

The best model resulted by Kaggle is SVM with a score of 0.58666.

Appendix

This section is left blank as no jargon was used for further clarification.