# Customer Churn Prediction

## List of Figures

# List of Tables

# Contents

# 1 Introduction

As part of the bank operations, some information regarding their customers are stored in the banking information systems. Typically, the bank uses the information they have about their customers in different decision-making processes such as for marketing purposes, credit card issuance, fraudulent transactions, etc. Due to different reasons, a customer may decide to stop banking with a particular bank. For example, some of these could be due to high charges on credit card facilities, exorbitant foreign transaction fees, account standard charges, etc.

The task in this study is to design and implement Machine Learning (ML) classification models (Novaković et al. 2017) that could help a bank to determine which customers are likely to withdraw from using their services or not. The dataset used in this study contains credit card information for the bank customers. The objectives of this study are enumerated below:

- Perform data cleaning and prepare the dataset for ML modelling;
- Model optimization by tuning the models' hyperparameters;
- Model evaluation;
- Avoiding model overfitting;
- Results analysis

## 1.1 Dataset and Goal

The dataset used contains credit card information about the bank's customers. If the bank customers decide to terminate their credit card contracts, the bank would record operating losses. Therefore, the goal of the study is to use ML models to classify the bank customers. This insight could help the bank to improve upon their services in order to prevent customer churn (Nie et al. 2011). In this study, we will use supervised ML models namely Random Forest Classifier (RFC)(Liu, Wang and Zhang 2012) and Support Vector Classifier (SVC) (Pisner and Schnyer 2020) to classify the bank's customers as an "Existing Customer" or an "Attrited Customer".

The dataset comprises of 6237 observations, 15 independent variables, and 1 target variable. The data dictionary is described below:

**Independent Variables**

- Customer_Age: Customer age in years
- Gender: M = Male, F = Female
- Dependent_count: Number of dependents
- Education_Level: Highest education level of the customer
- Marital_Status: Married, Single, Divorced
- Income_Category: Annual income category of the customer
- Card_Category: Type of card. Platinum cards offer more reward points than gold cards, which offers more reward points than silver cards. Blue cards offer the least number of reward points on purchases.
- Months_on_book: Period of relationship (in months) with bank.
- Total_Relationship_Count: Total number of products held by the customer (e.g., Saving account, Car loan, Credit Card, etc.)
- Months_Inactive: Number of months inactive in the last 12 months
- Contacts_Count: Number of customer service contacts in the last 12 months
- Credit_Limit: Credit limit on the credit card
- Total_Revolving_Bal: Total revolving balance on the credit card (the portion of credit card spending that goes unpaid at the end of a billing cycle)
- Total_Trans_Amt: Total transaction amount in the last 12 months
- Total_Trans_Ct: Total transaction count in the last 12 months

**Dependent / Target Variable**

- Attrition_Flag: Two labels - 'Existing Customer', 'Attrited Customer' (Customer who has churned)

## 2 Data Preparation

In this section, we describe the data preparation steps taken in this study. Figure 1 illustrates the typical workflow for ML systems. The process of applied ML comprises of a sequence of steps shown on the left-hand side (LHS).

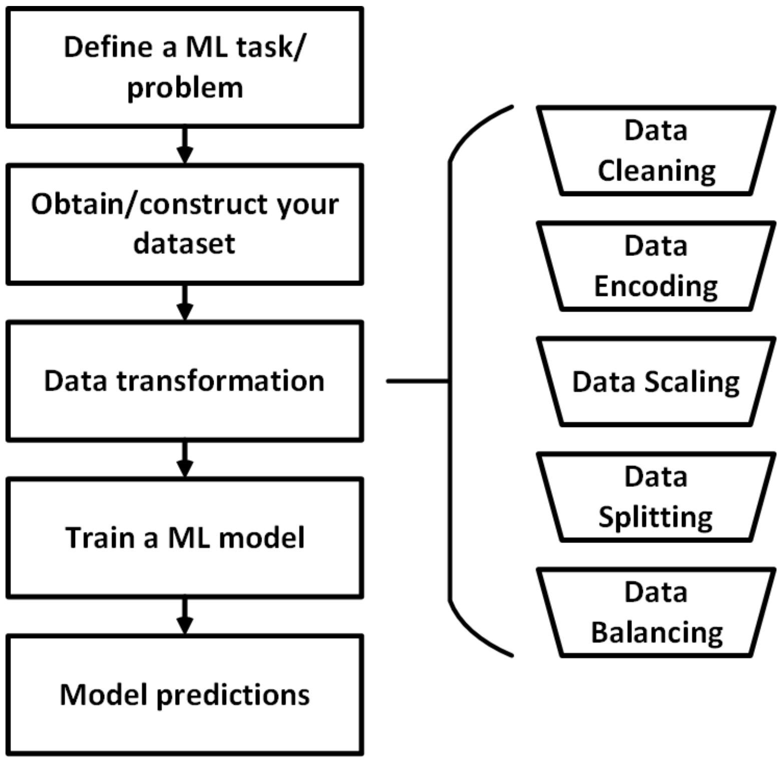


Figure 1: An overview of ML systems is shown (LHS). The RHS illustrates the data preparation process.

The first stage involves defining a ML problem. This step involves gathering sufficient project knowledge to choose articulate the ML task. This is where you decide if it is a classification problem, a regression problem, or another higher-order problem type (Simeone 2018). In this study, the task is a classification task where the objective is to determine if a customer churned or not. Next, the dataset is constructed or obtained. Some ML tasks may involve scraping the web (Kumar et al. 2021) to obtain the data. Data for ML projects can also be obtained from data repositories such as the UCI Machine Learning repository (*UCI Machine Learning Repository* n.d.).

The data preparation or transformation stage is where we process the raw data and make it ready for further ML processing and analysis. This step as shown in Figure 1 (RHS) starts with the cleaning the dataset. The other stages of the data preparation process includes data encoding, data scaling, data splitting and data balancing.

The data cleaning stage involves removing any NaN and fixing issues with missing numbers. A common practice is to perform data imputation (Emmanuel et al. 2021) where the mean of the column could be used in instances where there are missing entries. This is mostly useful in constrained datasets. Sometimes, this stage could involve data labeling or annotation.

In this study, we first start by checking which of the features in the dataset are continuous or categorical. This step is crucial as the categorical features would have to be encoded in order for us to apply ML models on this data. We begin importing the necessary libraries we would need to prepare the data. First, we read in the data and save it into a variable, 'data'.

```
import pandas as pd
import numpy as np
data = pd.read_csv('CustomerChurn.csv') #read in data
pd.set_option('display.max_columns', None)
```

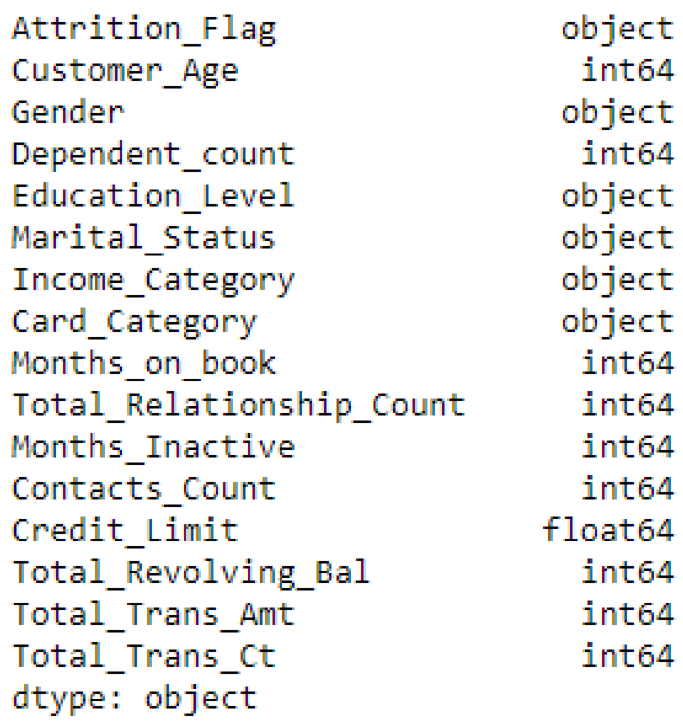
## 2.1 Handling the Continuous Features

In the code snippet below, we check the types of the features in the dataset. We observe that there are int64 (continuous) and categorical features (object type) in the dataset. The resulting output is shown in Figure 2.

```
    data.dtypes      #check for categorical and continuous variables
```

Next, we obtain a list of the categorical and continuous variables in the dataset.

```
    ''' Creating a list of the categorical features '''
obj_cols = []
for i in data:
     if data[i].dtypes == 'object':
```



```
Attrition_Flag                object
Customer_Age                   int64
Gender                        object
Dependent_count                int64
Education_Level               object
Marital_Status                object
Income_Category               object
Card_Category                 object
Months_on_book                 int64
Total_Relationship_Count       int64
Months_Inactive                int64
Contacts_Count                 int64
Credit_Limit                 float64
Total_Revolving_Bal            int64
Total_Trans_Amt                int64
Total_Trans_Ct                 int64
dtype: object
```

Figure 2: The feature types of the dataset is shown.

```
        obj_cols.append(i)
''' Creating a list of the continuous features '''
cont_cols = []
for i in data:
    if data[i].dtypes == 'int64' or data[i].dtypes == 'float64':
        cont_cols.append(i)
```

We need these list objects to check for missing (null) values and NaN in the continuous variables. The list object containing the categorical variables would be used to know the number of classes or levels in each categorical feature. The code snippet below illustrates the process of checking for NaN and null vales in the continuous variables.

```
for k in cont_cols:
    print(k, ":", data[k].isnull().sum())
for k in cont_cols:
    print(k, ":", data[k].isna().sum())
```

There were no missing values or NaN in the dataset.

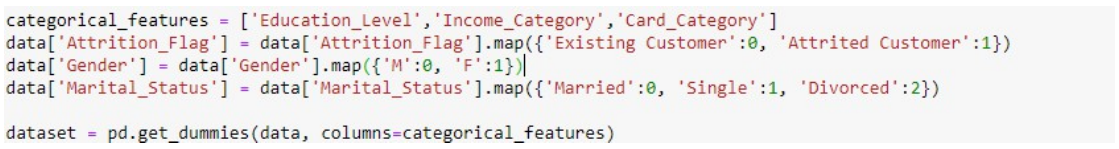
## 2.2 Encoding the Categorical Features

The dataset contains six categorical features that need to be converted to numerical values for the ML models. We start by checking each feature to know the number of classes in each variable for the encoding process (Garrido-Merchán and Hernández-Lobato 2020).

```
for k in obj_cols:
    print(k, ":", data[k].nunique())


# Output:
Attrition_Flag : 2
Gender : 2
Education_Level : 5
Marital_Status : 3
Income_Category : 5
Card_Category : 4
```

To encode the categorical features, we will use label encoding for features with 2 and 3 levels. These include the following features: Attrition_Flag, Gender and Marital_Status. The remaining categorical features, namely Education_Level, Income_Category and Card_Category will be encoded using the One-Hot encoding technique. This process is illustrated in Figure 3.



```
categorical_features = ['Education_Level','Income_Category','Card_Category']
data['Attrition_Flag'] = data['Attrition_Flag'].map({'Existing Customer':0, 'Attrited Customer':1})
data['Gender'] = data['Gender'].map({'M':0, 'F':1})
data['Marital_Status'] = data['Marital_Status'].map({'Married':0, 'Single':1, 'Divorced':2})

dataset = pd.get_dummies(data, columns=categorical_features)
```

Figure 3: One-hot encoding the categorical features.

## 2.3 Data Standardization, Splitting and Balancing

In this section, we describe the data standardization, balancing and spiltting procedures for the ML modelling. A common prerequisite for many ML models is the standardization of the dataset. This is done to prevent bias especially when comparing measurements with different units. This is because differently scaled features would not contribute fairly to the analysis. In this study, we use the Scikit-Learn StandardScaler (*Scikit-Learn StandardScaler* Accessed on 11/18/2022) technique to standardize the dataset. This method standardizes the feature set by removing the mean and scaling to unit variance.

```
X_scaled  =  StandardScaler().fit_transform(X.astype(float))
X_scaled = pd.DataFrame(X_scaled,columns=dataset_cols)
```

Next, we split the dataset into training and testing data using a 70% to 30% ratio. The training data consists of 70% of the dataset and will be used for training the ML models. The remaining 30% of the dataset was set aside for testing purposes. This testing data will be used to evaluate the performance of the ML algorithms.

In classification tasks, an imbalanced dataset is one with a skewed class proportion. The majority class is the class that makes up the larger proportion of the dataset. The smaller proportion of the dataset is the minority class. Looking at the entire dataset, we can see that the target class in the dataset is imbalanced as illustrated in Figure 4. The LHS of the diagram shows that the full dataset is dominated by the 'Existing Customer' level which accounts for 84.3%. The 'Attrited Customer' level accounts for the remaining 15.7%.

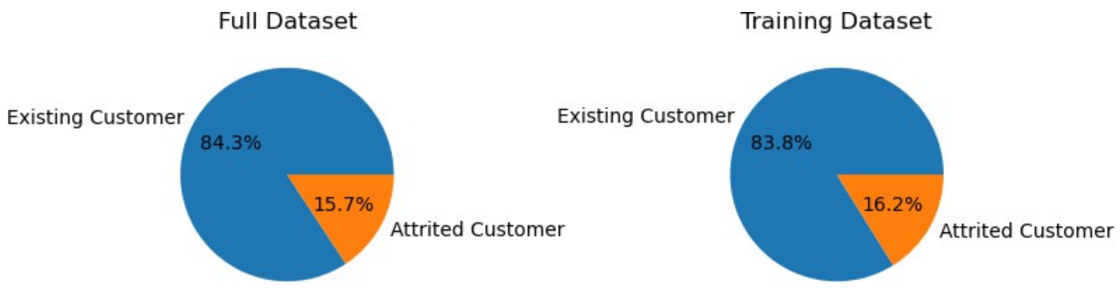
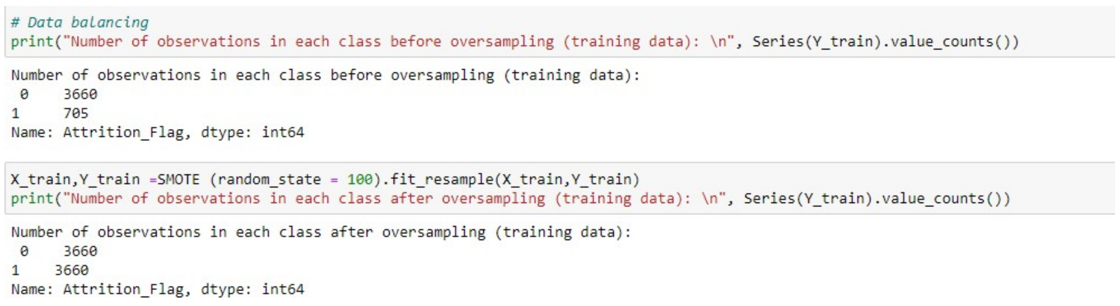


Figure 4: Class distribution for the target feature. The target class is unbalanced.

The RHS of Figure 4 illustrates that the training dataset is dominated by 'Existing Customer' level which accounts for 83.8%. The 'Attrited Customer' level accounts for the remaining 16.2%.

This imbalanced dataset could negatively impact the ML models if implemented this way. To address this, we applied Synthetic Minority Oversampling Technique (SMOTE) (Elreedy and Atiya 2019). Figure 5 shows the target class distribution before and after SMOTE implementation.



```
# Data balancing
print("Number of observations in each class before oversampling (training data): \n", Series(Y_train).value_counts())
```

```
Number of observations in each class before oversampling (training data):
 0    3660
1     705
Name: Attrition_Flag, dtype: int64
```

```
X_train,Y_train =SMOTE (random_state = 100).fit_resample(X_train,Y_train)
print("Number of observations in each class after oversampling (training data): \n", Series(Y_train).value_counts())
```

```
Number of observations in each class after oversampling (training data):
 0    3660
1    3660
Name: Attrition_Flag, dtype: int64
```

Figure 5: Data balancing using SMOTE.

## 3 Model Training

This section describes the model training procedure using the RFC and SVC classifiers. For both models, we investigate a range of scenarios which include the following: training the classification ML models using the default parameters, changing some parameters by intuition, using the most important features (RFC only), model hyperparameter tuning with *K*-fold cross-validation using the Scikit-Learn GridSearch method, and using the Scikit-Learn Pipeline technique.

### 3.1 Random Forest Algorithm

We use the random forest algorithm, which is an ensemble method. The random forest algorithm constructs multiple decision trees and combine their output in order to make stable and more accurate predictions (Breiman 2001).

### 3.1.1 RFC Model Training in Default Setting

We begin by fitting a basic RFC model in its default setting. We can see the default settings for a RFC object by using the code snippet below.

```
rfc_default = RandomForestClassifier(random_state=101)
print(rfc_default.get_params())
```

In this setting, we can see that the RFC object has the following parameters:

```
n_estimators=100, criterion='gini', max_features='auto'.
```

Here, we are training the RFC model using the default value of the number of trees for a RFC model which is 100. The function to decide the quality of the split is 'gini' and the maximum number of features to evaluate when looking for the ideal split has been set to 'auto'. By setting this to 'auto', then the maximum features to consider at each split will be the square root of the number of trees in the RFC algorithm,

```
max_features=sqrt(n_features).
```

### 3.1.2 Manually Chosen Parameters

In a second approach, we experiment by manually changing some of the default RFC algorithm parameters. Here, we train the RFC model with the following parameters:

```
n_estimators=50, criterion='entropy', max_features='auto'.
```

### 3.1.3 RFC Model with GridSearchCV

Next, we train the RFC model using the Scikit-Learn GridSearchCV. This function accepts a grid of parameter values and performs an exhaustive search over the specified values for the model estimators. With this function, we can fit our training data to the label and also predict on the testing data. We set the function to perform 5-fold cross validation. The method returns the 300 trees as the ideal number of trees for the model.

### 3.1.4 RFC Model with Most Important Features

Using the resulting model from the GridSearchCV method, we fit a RFC model using the top 12 most important features. Figure 6 illustrates a plot of the all features used in the RFC model and their significance.

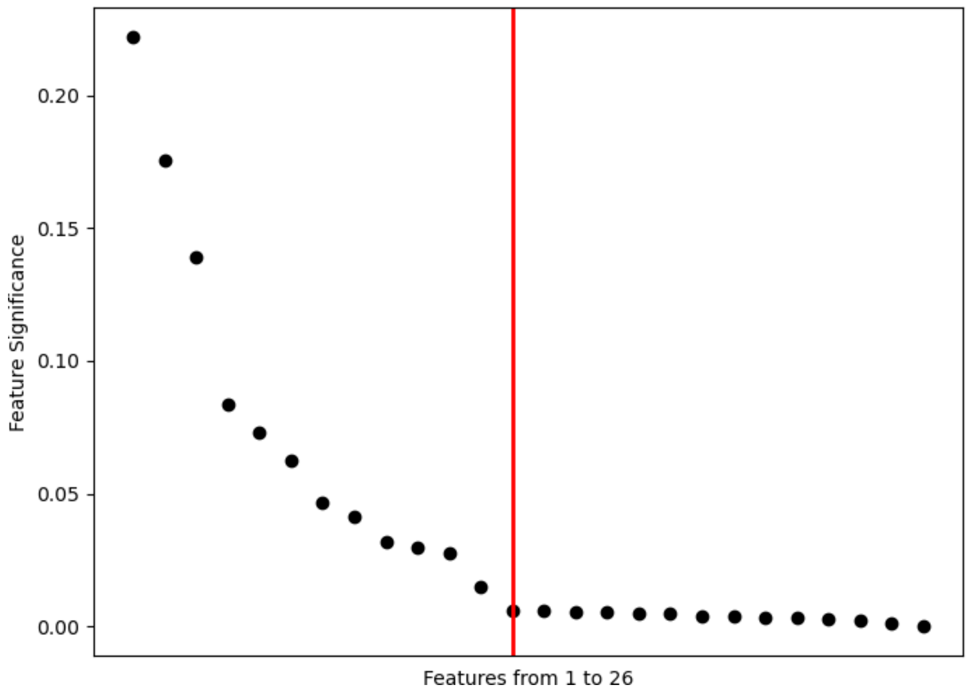


Figure 6: RFC Model Feature Importance.

Using plot above, we selected the top 12 features as we can see that there is a sharp decline in feature significance at this point. We set up a grid of parameter values (shown below) for this model training.

n_estimators=[64,100,128,200,250,300]
max_features= [2,4,6,8,10,12,'auto']

The GridSearchCV function using the top 12 most important features indicates that the ideal number of trees is 200 and maximum features to consider for splits should be 2.

Next, we evaluate the effect of excluding the max_features from the RFC model with the top 12 features. The model returns 200 as the ideal n_estimators parameter.

### 3.1.5 RFC Model using a Pipeline

We then evaluate the RFC model using a pipeline of transformers with a final estimator. The goal of using a pipeline is to put together a number of phases that can be cross-validated while using various parameters. We set the following parameters for the pipeline:

params = {'classification_n_estimators':[100,200,250,300,350,400], 'classification___max_features':[2,4,6,8,10,12,'auto']}.

The model returns the following ideal parameters:

'classification__max_features': 10, 'classification__n_estimators': 350.

Next, we train a RFC model using a pipeline with the top 12 important features. This time the model returns the best RFC model parameters as shown below:

'classification_max_features': 2, 'classification_n_estimators': 300.

## 3.2 Support Vector Machine Algorithm

In comparison with some other types of ML classifiers such as the decision trees, support vector machines offer better performance and accuracy. Due to its kernel method, the algorithm is able to handle non-linear feature space.

### 3.2.1 SVC Model Training in Default Setting

We begin by fitting a basic SVC model in its default setting. Some important model parameters in the default setting are shown below:

'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf'

The C parameter controls the regularization of the model, kernel specifies the kernel type and the gamma parameter indicates the kernel coefficient.

### 3.2.2 Manually Chosen Parameters

In a second approach, we investigate the SVC model performance by manually changing some of the default SVC algorithm parameters. Here, we train the SVC model with the following parameters:

```
kernel='linear', gamma='auto', C=10.
```

### 3.2.3 SVC model using a Pipeline

We then evaluate the SVC model using a pipeline of transforms with a final estimator. To do this, we use a grid of parameters (shown below) and a range of values and a 5-fold cross-validation.

```
kernels_c = {'classification__kernel': ['linear','poly','rbf','sigmoid'],
'classification__C': [.001,.01,.1,1,10,100]}
```

The search returns the following as the best parameters for the SVC model:

```
'classification_C': 100, 'classification_kernel': 'rbf'
```

## 4 Model Hyperparameter Tuning

The RFC and the SVC models come bundled with a range of hyperparameters that can be tuned to optimize the model performance. For each model, Scikit-Learn implements a set of reasonable default hyperparameters, but there is no guarantee that these are the best ones for the given situation. For example, to see the default parameters for a SVC model, one can run the following lines of code:

```
from sklearn.svm import SVC
svc_default = SVC(random_state=100)
print(svc_default.get_params()).
```

### 4.1 Cross Validation

To tune the models' hyperparameters, we adopted a 5-fold cross validation technique. A 5-fold CV would split the training dataset into five subsets, called folds.

In using a 5-fold CV, we train the models using the first four folds and evaluate on the fifth for the first iteration. During the second iteration, we train the models using first, second, third, and fifth folds and evaluate the model performance on the fourth. This process is then repeated three more times, assessing the model performance each time on a different subset or fold. The performance on each of the folds is averaged at the very end of training to get the model's final validation metrics.

## 4.2 RFC Hyperparameter Tuning

The random forest algorithm when compared with a decision tree has four unique parameters that are not present in a decision tree model. These parameters are:

```
{'n_estimators': 100, 'max_features': 'auto',

'bootstrap': True, 'oob_score': False}.
```

In our model evaluations, we focused on the n_features and the max_features. This is because the n_features parameters indicates the number of decision trees we would use in the RFC model. This parameter combined with the 'bootstrap' parameter decides how many feature subsets and data rows are used in the RFC model. Boostrapping, which by default is set to 'True' is a hyperparaemeter meant to reduce the correlation between trees. This is because the trees are trained on different subsets of the feature columns and observations. This helps ensure that the RFC model will generalize well to new data.

The other hyperparameter we tuned for the RFC models was the max_features. This option indicates the number of features the RFC model should use in each of the random subsets at each split. When set to 'auto', the model will use the square root of the number of trees in the RFC model.

## 4.3 SVC Hyperparameter Tuning

In optimizing the performance of the SVC model, we focused on two parameters: the kernel and the C parameters. This is because the kernel function is used to transform the training data so that a non-linear decision layer can be transformed into a linear equation. The main objective of using the kernel is to re-map a low-dimensional input space into a higher-dimensional space.

The other parameter we tuned for the SVC model is the C parameter. This parameter represents the regularization parameter. This is the misclassification or error term of the SVC model. It helps control trade-off between the decision boundary and the misclassification rate.

## 5 Model Evaluation Metric

The model classification accuracies are evaluated with the accuracy metric derived from the models' classification confusion matrix. The accuracy metric is

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}. \tag{1}$$

where TP and FP represent True and False Positives; TN and FN represent True and False Negatives.

For the problem domain we consider in this study, the accuracy of the model would not be an ideal evaluation metric to adopt.

**A model can make a wrong prediction:**

1. Predicting that a customer will not attrite whereas the customer actually attrites. There is a potential loss of opportunity for the bank to do something differently that may prevent this from happening again.
2. Predicting that a customer will attrite whereas the customer does not in reality. This presents a loss of resources that may have been channelled wrongly to prevent attrition.

In addition to model accuracies, classification models can be evaluated using precision and recall. The precision of a model evaluates how well a model predicts positive instances which are indeed positive. In other words, precision shows how much we can trust that our model is accurate when predicting that a customer will not attrite and the customer does not attrite in reality. The precision of a model is computed as

$$Precision = \frac{TP}{TP + FP}. \qquad (2)$$

Recall is a metric that evaluates the ability of the model to avoid false negatives. The recall metric is calculated by dividing the number of *TP* elements by the total number of positively classified units (Grandini, Bagli and Visani 2020). The recall of a model is computed as

$$Recall = \frac{TP}{TP + FN}. \qquad (3)$$

### 5.1 Choice of Evaluation Metric

The preferred choice of evaluation metric for this study is the Recall metric. This is because a model prediction that a customer will not attrite in a case that the customer does actually attrite will lead to a potential loss for the bank. The bank marketing, sales or customer relationship team could have done things differently or made contacts with the customer to retain them if the model had predicted this correctly. Losses would ensue from this. So, it is important to reduce false negatives. Therefore, the bank would want to maximize Recall because higher Recall rates reduces the likelihood of false negative results (Routh, Roy and Meyer 2021).

## 6 Overfitting Avoidance Mechanism

When creating a predictive model, the process of feature selection involves reducing the number of input variables. In some circumstances, reducing the number of input variables might enhance the efficiency of the model while also lowering the computational costs. Other techniques to prevent model overfitting include regularization, ensembling, data augmentation and training with more data (Ying 2019).

### 6.1 RFC: Feature Selection and Cross Validation

In this study, to avoid overfitting with the RFC models, we adopted feature selection. In Figure 6, we presented a plot showing the features used in the RFC and their relative significance. We then selected the top 12 most important variables by visually inspecting the plot for a spot with a sharp decline in significance. By selecting only the top 12 features from the feature space, we eliminated unimportant variables and boosted the models classification performance (Chen et al. 2020). Secondly, using a 5-fold cross validation when training the RFC model was another mechanism adopted to prevent overfitting.

### 6.2 SVC: Regularization and Cross Validation

To avoiding overfitting with the SVC models, we applied a 5-fold cross validation and tuned the C parameter. The C parameter in the SVC model controls how much you want to avoid misclassifying the individual observation in the training data. We used a grid of C parameter values with the GridSearchCV to choose the best model parameters for the SVC model. Larger values of C, the SVC optimization will pick a smaller-margin hyperplane if it performs better at accurately classifying every training observation. Smaller values of C will result to the larger-margin hyperplanes. Even if your training data can be linearly separated, you should frequently encounter misclassified cases for extremely low values of C (Ben-Hur and Weston 2010).

## 7 Results Analysis

Table 1 lists the prediction results of the RFC model using the testing data. The classification metrics are reported in percentages. The RFC model in its default setting produces the worst Recall performance for all scenarios investigated. The model records approximately 78%, 81% and 94% for Recall, Precision and Accuracy respectively.

The RFC model produces the best Recall and Accuracy metrics using a Pipeline with the top 12 most important features. The Recall and Accuracy of this model

Table 1: The prediction results of the RFC model classification using the testing data are listed. The results are in (%). The best Recall metric was achieved with the RFC model using the Pipeline with the top 12 features.

| | | Testing Dataset Results | | |
|---|---|---|---|---|
| RFC Model | CV | *Accuracy* | *Recall* | *Precision* |
| Default Settings | No | 94.18 | 79.71 | 80.59 |
| Manual Parameters | No | 94.28 | 80.43 | 80.73 |
| GridSearchCV | 5-fold | 94.44 | 88.88 | 80.71 |
| Top 12 Features | 5-fold | 95.35 | 86.67 | 83.45 |
| Pipeline | 5-fold | 94.02 | 82.61 | 78.08 |
| Pipeline with Top 12 | 5-fold | 95.41 | 87.37 | 83.28 |

is 87.37% and 95.41% respectively. Comparing the model's Recall with the RFC model using the Scikit-Learn default parameters, this is approximately a 9.6% improvement in the Recall classification metrics.

The SVC prediction results using the testing data are listed in Table 2. The SVC model in its Scikit-Learn default settings produces a Recall of 76.81%. The Accuracy and Precision for this model are approximately 87% and 55% respectively. The SVC produces the best Recall of 82% with the randomly chosen parameters. From the results, we observe that the SVC performance drops to 56% with a Pipeline using the GridSearchCV. Apparently, this is a good indicator that the model seems to be overfitting.

## 7.1 Recommendation

Based on the results from both the models, the RFC and SVC algorithms, the recommendation for real-world deployment would be the RFC algorithm. This is because for the dataset used in this study, the ensemble technique of the RFC produces better classification results compared with the SVC models. RFC models are made up of many decision trees, each one trained on a random bootstrapped sample of the characteristics and observations. Each tree becomes roughly decor-

Table 2: The prediction results of the SVC model classification using the testing data are listed. The results are in (%). The best Recall metric was achieved with the SVC model using the manually chosen parameters.

| | | Testing Dataset Results | | |
|---|---|---|---|---|
| RFC Model | CV | *Accuracy* | *Recall* | *Precision* |
| Default Settings | No | 87.34 | 76.81 | 55.06 |
| Manual Parameters | No | 82.91 | 82.25 | 45.58 |
| GridSearchCV | 5-fold | 87.23 | 56.16 | 56.78 |

related in this way. A consensus vote is then collected for classification or a mean for regression for predictions. The variance of the predictions is reduced by the bagging technique used by combining the random trees in the forest.

The SVC models seemed to be overfitting judging from the results in Table 2. The SVC performed poorly when presented with the testing data. It failed to generalize well to these unseen data. When using the SVC models, selecting the optimal kernel for the dataset is crucial. From the SVC models tabulated in Table 2, one may wrongly conclude that the models are good juding from the model accuracies. The Recall and Precision metrics demonstrate why Accuracy may not always be an ideal metric. This is true especially for a study like this where the goal is to maximize the Recall metrics.

## 7.2 Conclusions

Considering a real-world deployment, the RFC models proposed in this study can enable the bank to identify the customers who are most likely to leave. Then, they can channel their marketing, sales and customer relationship efforts to this group of customers. The bank can decide to reach out to the top percentage of those customers to talk about credit card offers, credit limit increases, and other ways to attempt to keep them from leaving.

## References

Ben-Hur, Asa and Jason Weston (2010). ‘A user’s guide to support vector machines’. In: *Data mining techniques for the life sciences*. Springer, pp. 223–239.

Breiman, Leo (2001). ‘Random forests’. In: *Machine learning* 45.1, pp. 5–32.

Chen, Rung-Ching et al. (2020). ‘Selecting critical features for data classification based on machine learning methods’. In: *Journal of Big Data* 7.1, pp. 1–26.

Elreedy, Dina and Amir F Atiya (2019). ‘A comprehensive analysis of synthetic minority oversampling technique (SMOTE) for handling class imbalance’. In: *Information Sciences* 505, pp. 32–64.

Emmanuel, Tlamelo et al. (2021). ‘A survey on missing data in machine learning’. In: *Journal of Big Data* 8.1, pp. 1–37.

Garrido-Merchán, Eduardo C and Daniel Hernández-Lobato (2020). ‘Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes’. In: *Neurocomputing* 380, pp. 20–35.

Grandini, Margherita, Enrico Bagli and Giorgio Visani (2020). ‘Metrics for multi-class classification: an overview’. In: *arXiv preprint arXiv:2008.05756*.

Kumar, Swarn Avinash et al. (2021). ‘A machine-learning scraping tool for data fusion in the analysis of sentiments about pandemics for supporting business decisions with human-centric AI explanations’. In: *PeerJ Computer Science* 7, e713.

Liu, Yanli, Yourong Wang and Jian Zhang (2012). ‘New machine learning algorithm: Random forest’. In: *International Conference on Information Computing and Applications*. Springer, pp. 246–252.

Nie, Guangli et al. (2011). ‘Credit card churn forecasting by logistic regression and decision tree’. In: *Expert Systems with Applications* 38.12. ISBN: 0957-4174 Publisher: Elsevier, pp. 15273–15285.

Novaković, Jasmina Dj et al. (2017). ‘Evaluation of classification models in machine learning’. In: *Theory and Applications of Mathematics & Computer Science* 7.1. Publisher: " Aurel Vlaicu" University of Arad Department of Mathematics and Computer . . ., p. 39.

Pisner, Derek A. and David M. Schnyer (2020). ‘Support vector machine’. In: *Machine learning*. Elsevier, pp. 101–121.

Routh, Pallav, Arkajyoti Roy and Jeff Meyer (2021). 'Estimating customer churn under competing risks'. In: *Journal of the Operational Research Society* 72.5, pp. 1138–1155.

*Scikit-Learn StandardScaler* (Accessed on 11/18/2022). https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

Simeone, Osvaldo (2018). 'A very brief introduction to machine learning with applications to communication systems'. In: *IEEE Transactions on Cognitive Communications and Networking* 4.4, pp. 648–664.

*UCI Machine Learning Repository* (n.d.). https://archive.ics.uci.edu/ml/index.php. (Accessed on 11/18/2022).

Ying, Xue (2019). 'An overview of overfitting and its solutions'. In: *Journal of physics: Conference series*. Vol. 1168. 2. IOP Publishing, p. 022022.