HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT
BANK CUSTOMER CHURN

APPLIED STATISTIC AND EXPERIMENTAL DESIGN

LECTURE:

NGUYEN LINH GIANG

GROUP 18:

Do Nghiem Duc 20214892

Nguyen Huu Nam 20210630

Hoang Tu Quyen 20214929

Pham Duc Thanh 20210795

Tran Van Toan 20214932

ABSTRACT

This report presents a customer bank churn project that aims to predict customer churn in the banking industry. The project utilizes several machine learning algorithms, including k-Nearest Neighbors (k-NN), Random Forest, Support Vector Machine (SVM), Bayesian Network, and Naive Bayes. The report outlines the significance of customer churn in banking, describes the methodology used, and discusses the results and evaluation metrics of each algorithm. Key features and variables contributing to churn behavior are identified. The report concludes with recommendations for proactive retention strategies and further research opportunities.

Table of Contents

I. Introduction

1. Context

Every day there is much competition growing in the banking industry. Thus, if any bank wants to increase its market share by acquiring new customers, it must follow customer retention strategies. It is shown that improving the retention rate by up to 5% can increase a bank's profit by up to 85% [2]. However, there will be multiple opportunities for clients to churn out of a particular bank for every customer who has more than one credit card with more than one bank. Whenever a customer realizes that Bank A offers many facilities at a low-interest rate compared to Bank B, the customer churning prediction for Bank B is high. Therefore, it is the bank credit card account management system responsibility to ensure that the existing customers are maintained through low interest rates. Nowadays, with the intense machine learning advancement, it is beneficial to build a prediction approach that able to predict whether a credit cardholder or a customer will churn out from a particular bank or not.

For the project, we collect credit card churn customer data of around 10,000 from Kaggle repository.

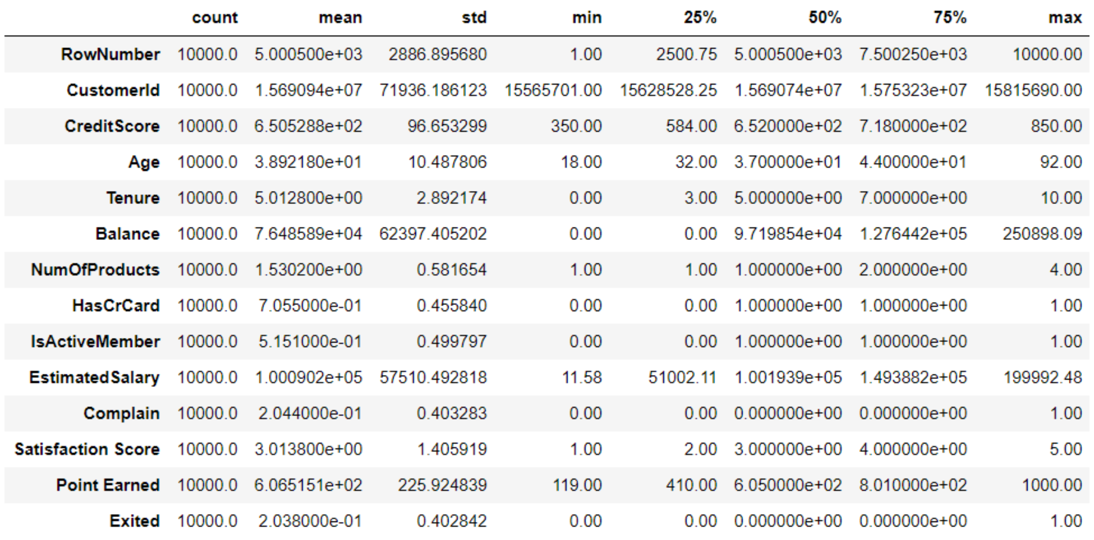
2. Attribute Information

RowNumber	The identifier assigned to each row
CustomerId	The uniquely identifies each customer
Surname	The customer's last name or surname
CreditScore	The creditworthiness of a customer
Geography	The customer's geographical location [France, Spain, Germany]
Gender	The customer's gender [M: Male, F: Female]
Age	The customer's age
Tenure	The length of time the customer has been associated with the bank
Balance	The customer's account balance
NumOfProducts	The number of products or services the customer has with the bank
HasCrCard	The customer has a credit card with the bank (1) or not (0)
IsActiveMember	The customer is an active member (1) or not (0)
EstimatedSalary	The estimated salary of the customer
Satisfaction Score	The level of satisfaction expressed by the customer
Exited	The customer has churned (1) or not (0)
Complain	The customer has lodged a complaint with the bank
Card Type	The type of credit card held by the customer [DIAMOND, GOLD, SILVER, PLATINUM]
Point Earned	The loyalty points or rewards earned by the customer

II. Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is one of the most important steps before doing anything in every tabular dataset problem. As we used Bank Customer Churn Prediction Dataset, EDA helps us gain insight into the dataset by visualizing charts and detecting any errors, outliers as well as understanding different data patterns of attributes. In this section, we tried o depict what we have done and how we preprocessed the dataset.

1. Data Insight and Visualization:



	count	mean	std	min	25%	50%	75%	max
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.500250e+03	10000.00
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.575323e+07	15815690.00
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.180000e+02	850.00
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.400000e+01	92.00
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.000000e+00	10.00
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.276442e+05	250898.09
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.000000e+00	4.00
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.000000e+00	1.00
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.000000e+00	1.00
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.493882e+05	199992.48
Complain	10000.0	2.044000e-01	0.403283	0.00	0.00	0.000000e+00	0.000000e+00	1.00
Satisfaction Score	10000.0	3.013800e+00	1.405919	1.00	2.00	3.000000e+00	4.000000e+00	5.00
Point Earned	10000.0	6.065151e+02	225.924839	119.00	410.00	6.050000e+02	8.010000e+02	1000.00
Exited	10000.0	2.038000e-01	0.402842	0.00	0.00	0.000000e+00	0.000000e+00	1.00

Figure 1. Numerical Value Information

Upon examining the Numerical Value Information (Figure 1), we observed that RowNumber, CustomerId, CreditScore features are not directly related to churn prediction. Thus, we decide to remove these features.

Next, we would like to remove redundant attributes and some attributes which have a high correlation value with other attributes which are based on a heatmap (Figure 2).

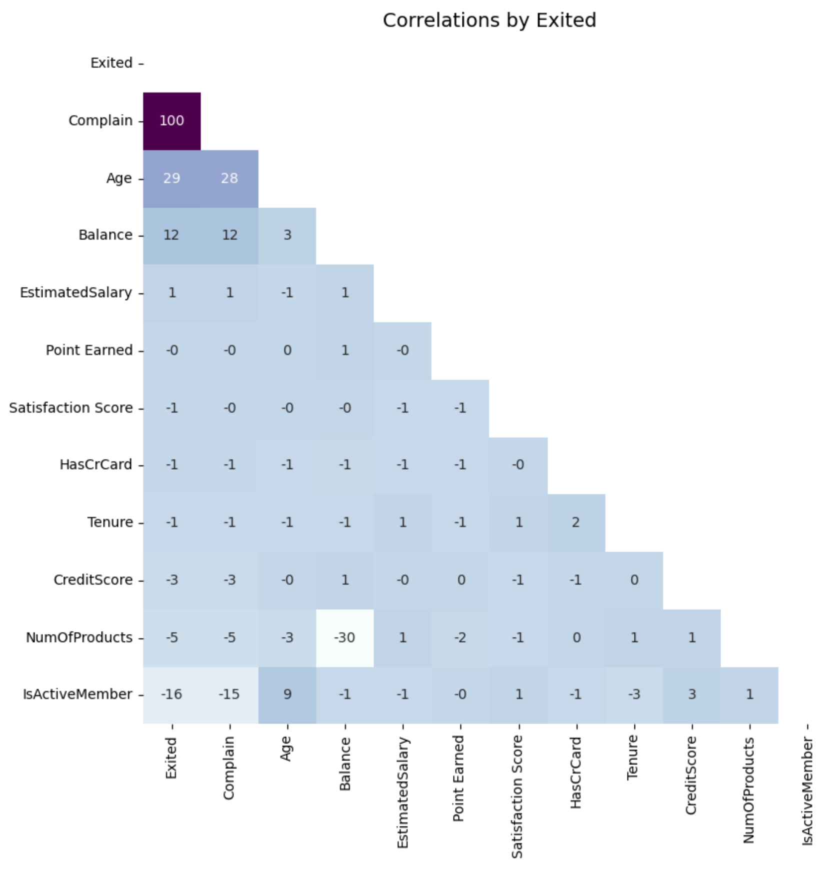


Figure 2. Correlation heatmap

Due to the high correlation between the "Complain" variable and the "Exited" variable, we have decided to drop the "Complain" variable in order to avoid potential issues of multicollinearity and enhance the model's performance in predicting customer churn.

Next, we create a visualization table to identify null values or any anomalous values.

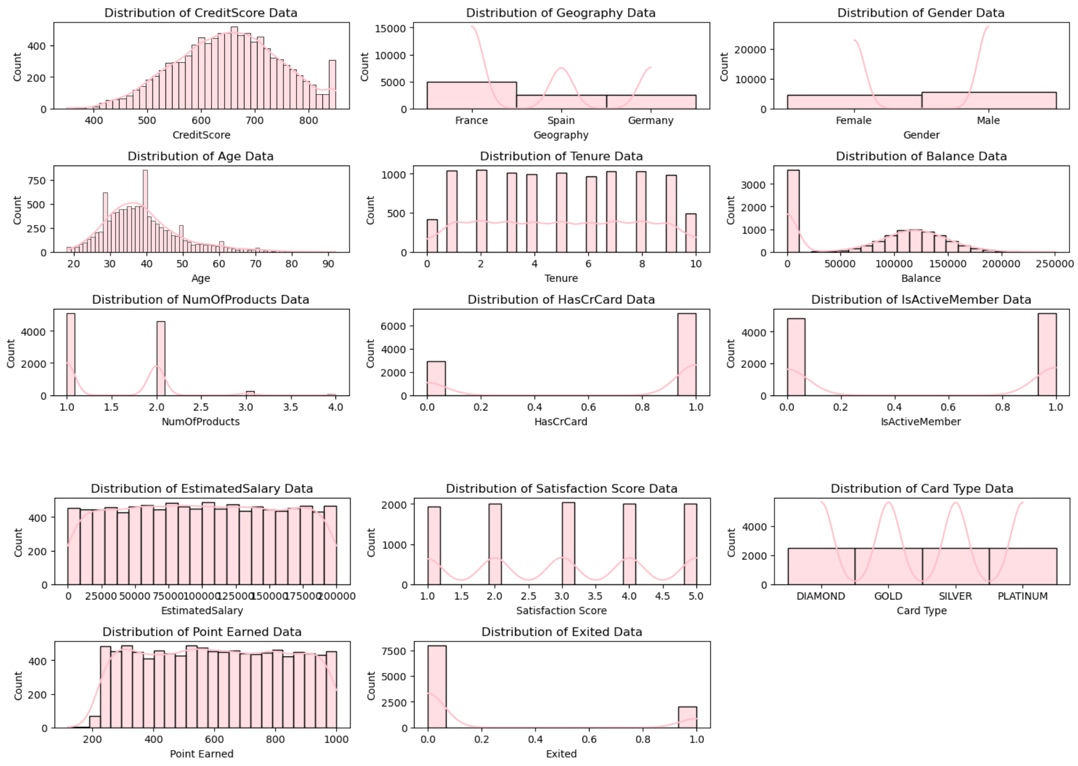


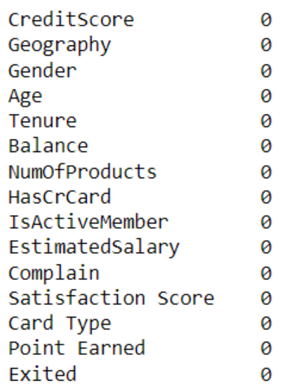
Figure 3. Data visualization for every attributes

2. Data Preparation

Data preparation is a pre-processing step that involves cleansing, transforming, and consolidating data. In this part, we will illustrate how we preprocessed our dataset based on the previous section.

2.1. Data cleansing

During this step, we conducted a thorough examination to identify any missing or inappropriate values that deviated significantly from the expected descriptions for each attribute (Figure 4).



```
CreditScore          0
Geography            0
Gender               0
Age                  0
Tenure               0
Balance              0
NumOfProducts        0
HasCrCard            0
IsActiveMember       0
EstimatedSalary      0
Complain             0
Satisfaction Score   0
Card Type            0
Point Earned         0
Exited               0
```

Figure 4. Check null value

2.2. Analysis the target features

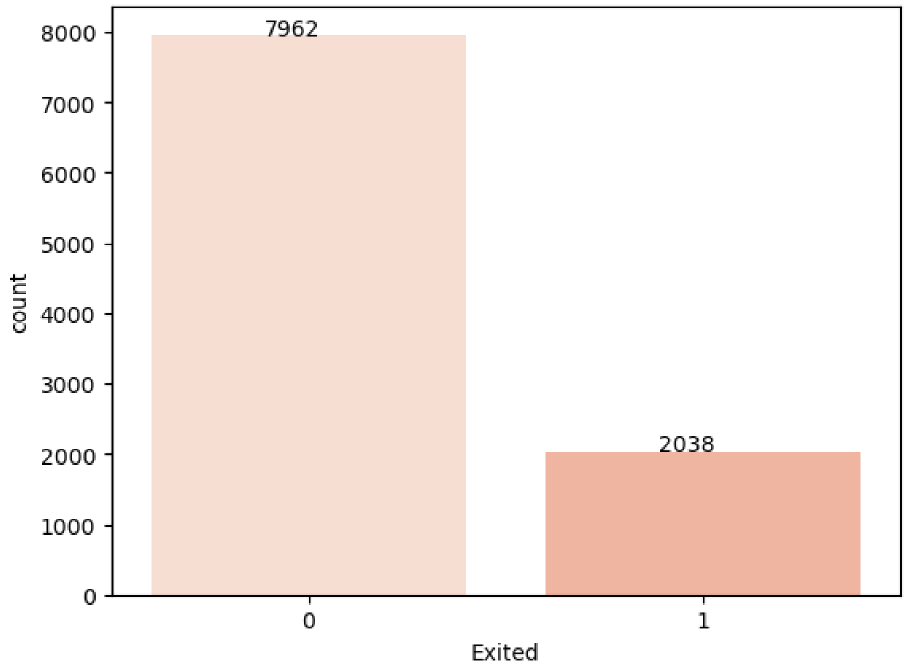


Figure 5. Target feature

We can see that our dataset is imbalanced. The majority class, "Stays" (0), has around 80% data points and the minority class, "Exits" (1), has around 20% datapoints.

To address this, in our machine learning algorithms we will use SMOTE (Synthetic Minority Over-sampling Technique).

The synthetic minority oversampling technique (SMOTE) can be described as a statistical technique. This technique aims to increase the number of cases in our dataset in a balanced manner. We generate new instances from our existing minority cases to feed our model. In this way, new instances are not simply copies of existing minority cases; instead, the algorithm takes a sample of the feature space for each target class and its nearest neighbors and creates new examples that combine features of the target case and those of its neighbors. The new approach increases the number of features available to each class and makes the samples more general. In order to increase the percentage of minority cases that are not attracted customers to twice the rate of majority cases, we use SMOTE.

2.3. Variable Transformations

Machine learning models do not understand the units of the values of the features. It treats the input just as a simple number but does not understand the true meaning of that value. Thus, it becomes necessary to scale the data. We use scaling for numerical features and encoding for categorical features.

2.3.1. Scaling

Since all the features have the data which does not display normal distribution so we decide to use MinMax Scale for all the numerical features.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2.3.2. Encoding

For categorical columns, we use one-hot encoding for all categorical features.

3. Data Splitting

We randomly split the dataset into 2 parts: train set and test set with the ratio 80%:20%

III. Modelling

1. Support Vector Machines

Kernel Support Vector Machine

Problem: Data are not linearly separable in 2D so we need to transform them to a higher dimension where they will be linearly separable. So we use the Kernel Support Vector Machine to solve this problem. In deed, there are 4 popular types of Kernel SVM: Linear, Polynomial, Radial Basis Function(RBF), Sigmoid. Our approach only considers RBF, since in general, the RBF (Radial Basis Function) kernel is the better choice when the data is not linearly separable and has a complex pattern (it is also sklearn's default kernel). Its formula:

$K(x,z) = e^{\frac{\|x-z\|^2}{2\sigma}}$, where $\sigma > 0$

Training Classifier: The RBF kernel has two important parameters: gamma and C (also called regularization parameter):

- Gamma is a parameter that determines the width of the kernel function.
- C is a regularization parameter that controls the trade-off between achieving a good fit to the training data and a simple decision boundary.
- Gamma controls the distance of influence of a single training point. The behavior(shape) of the model is very sensitive to the gamma parameter (Figure 17).

If gamma is too large, every point needs to be very close to each other to be considered as a group. So the radius of the area of influence of the support vectors only includes the support vector itself. The model too fits to training data and no amount of regularization with C will be able to prevent overfitting.

When gamma is very small, more points will be grouped together even if they are far away from each other. So the model cannot capture the complexity or "shape" of the data.

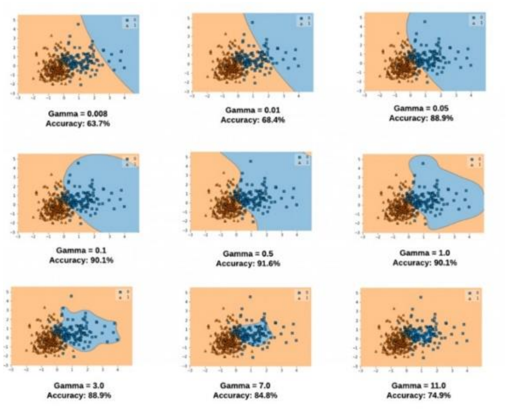
The C parameter is a regularization parameter used to set the tolerance of the model to allow the misclassification of data points in order to achieve lower generalization error. There is a tradeoff between "narrow cushion, little / no mistakes" and "wide cushion, quite a few mistakes" and the C value controls the penalty of misclassification (Figure 17).

If C is small, penalties for misclassified points are low so the decision boundary can have a large margin, therefore a simpler decision function, at the cost of training accuracy.

If C is large, SVM tries to minimize the number of misclassified points due to high penalty which result in decision boundary with smaller margin, thus forcing the model to be stricter to the training data and potentially overfit.

The effect of Gamma and C on the variance and bias is demonstrated in Figure 18.

In order to train parameters: gamma and C, we use 5-folds cross validation and grid search to find optimal values of gamma and C. After testing and training data, it seems that the best combination is gamma=0.01 and C=150 (the plot in the middle of Figure **17**).



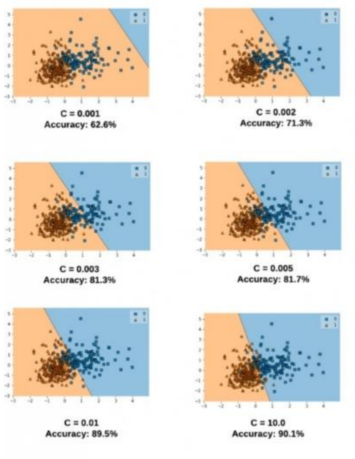


Figure 17: Effect of Gamma and C on SVM model

	Large Gamma	Small Gamma	Large C	Small C
Variance	Low	High	High	Low
Bias	High	Low	Low	High

Figure 18: Effect of Gamma and C on model's Variance and Bias

2. Neural networks

The use of multilayer perceptron (MLP)

2.1. Architecture

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 32)                288

 dropout_4 (Dropout)         (None, 32)                0

 layer_normalization_4 (Laye  (None, 32)               64
 rNormalization)

 dense_11 (Dense)            (None, 32)                1056

 dropout_5 (Dropout)         (None, 32)                0

 layer_normalization_5 (Laye  (None, 32)               64
 rNormalization)

 dense_12 (Dense)            (None, 32)                1056

 dense_13 (Dense)            (None, 1)                 33

=================================================================
Total params: 2,561
Trainable params: 2,561
Non-trainable params: 0
```

Architecture

Input Layer: It expects input data of shape (8,) which means it takes a vector of 8 features.

3 Dense Layers: This layer has 32 neurons and uses the $ReLU$ activation function. It takes the input from the previous layer.

$$\text{ReLU}(x) = \max(0, x)$$

2 Dropout Layers: It randomly sets 10% of the input units to 0 at each update during training, which helps prevent overfitting.

2 Normalization Layers: It normalizes the activations of the previous layer, improving stability and performance.

Output Layer: The final layer with 1 neuron and sigmoid activation function, which outputs a value between 0 and 1 representing the predicted probability of the binary class.

$$sigmoid(x) = 1 / (1 + e^{-x})$$

2.2. Optimizer

The formulas for the *Adam optimizer*:

Compute the gradient of the loss function with respect to the parameter:

$$g(t) = \nabla\big(L(\theta)\big)$$

Momentum:

$$v(t) = \beta_1 * v(t-1) + (1-\beta_1) * g(t)$$

Adaptive learning rate:

$$s(t) = \beta_2 * s(t-1) + (1-\beta_2) * \big(g(t)\big)^2$$

Update the parameter (including bias-corrected and a small constant to avoid division by zero):

$$\theta(t) = \theta(t-1) - \frac{\eta}{\left[sqrt\big(s(t)/(1-\beta 2^t)\big) + \varepsilon\right]} * \left[\frac{v(t)}{(1-\beta 1^t)}\right]$$

2.3. Loss

The formula for binary cross entropy loss:

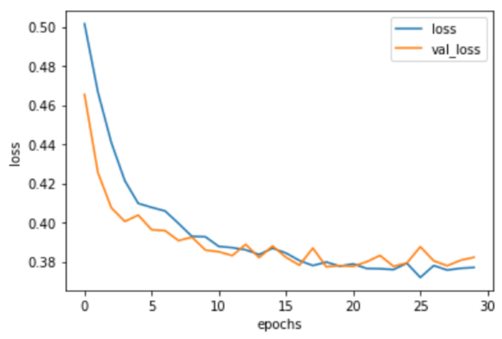
$$L(y, \hat{y}) = -\big(y * log(\hat{y}) + (1-y) * log(1-\hat{y})\big)$$

Where:

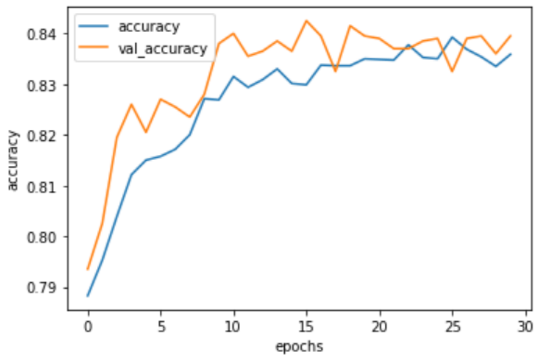
- y is the true binary label (0 or 1).
- ŷ is the predicted probability of the positive class (between 0 and 1) obtained from the model.

2.4. Result

Loss and accuracy after 30 epochs with mini batch size 32:



Loss



Accuracy

3. k-Nearest Neighbors Classifier

k-Nearest Neighbors Classifier algorithm assumes all instances correspond to points in the n-dimensional space $\mathfrak{R}^n$. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. More precisely, let an arbitrary instance x be described by the feature $\langle a_1(x), a_2(x), \ldots, a_3(x)\rangle$ vector.
where $a_r(x)$ denotes the value of the rth attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

$$d\left(x_i, x_i\right) \equiv \sqrt{\sum_{r=1}^{n}\left(a_r\left(x_i\right) - a_r\left(x_j\right)\right)^2}$$

In k-nearest neighbor learning the target function may be either discrete-valued or real-value.

3.1. Features Selection

Permutation importance is a technique that measures the importance of each feature by shuffling the values of that feature while keeping other features unchanged and observing the impact on the model's performance. The higher the decrease in performance after shuffling, the more important the feature is considered. The importance scores are calculated based on the accuracy metric, which reflects the model's ability to correctly predict customer churn.

From the feature importance scores, the top 5 features with the highest scores are selected for further analysis. These features have demonstrated the most significant impact on the model's predictive performance. The selected features are ranked based on their scores in descending order.

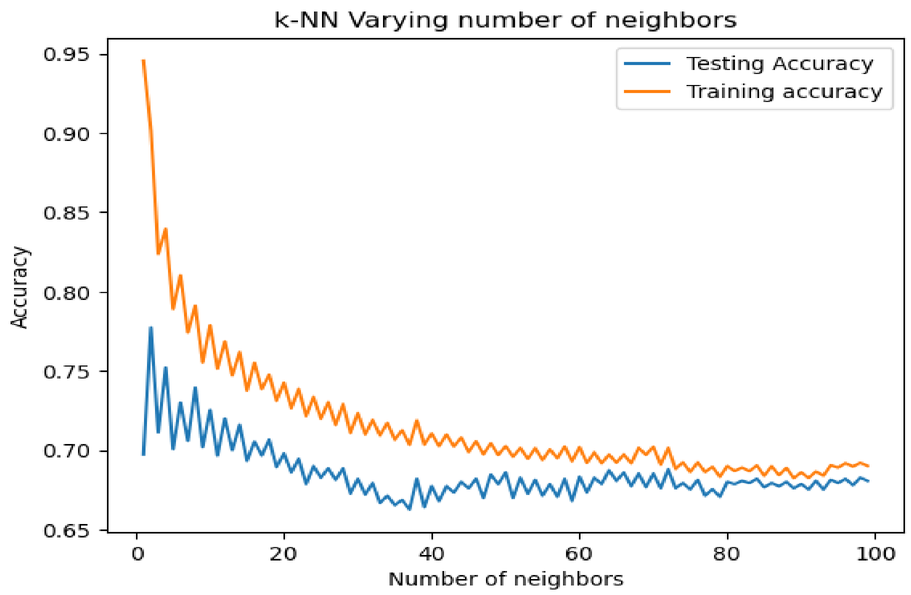
3.2. Fine Tuning with Grid-SearchCV

To further optimize the performance of the KNN classifier model in our bank customer churn project, a fine-tuning technique called GridSearchCV was employed. GridSearchCV is a powerful method for systematically exploring different combinations of hyperparameters and determining the optimal configuration for the model.

3.3. Evaluating Model Performance

To further investigate the impact of the number of neighbors on the performance of the KNN classifier in our bank customer churn project, we conducted an analysis using varying values of K. The aim was to identify the optimal number of neighbors that yields the highest accuracy for predicting customer churn.

We utilized the k_neighbors array, ranging from 1 to 100, to iterate through different values of KBy examining the plot, we can identify the optimal number of neighbors that yields the highest accuracy. The point on the graph where the testing accuracy curve peaks indicates the number of neighbors that provides the best balance between bias and variance.



The evaluation

```
              precision    recall  f1-score   support

         0.0       0.88      0.76      0.82      1207
         1.0       0.37      0.59      0.46       293

    accuracy                           0.73      1500
   macro avg       0.63      0.67      0.64      1500
weighted avg       0.78      0.73      0.75      1500
```

4. Decision tree

Decision tree is a tree-like model that represents decisions and their potential consequences in a flowchart-like structure. Each internal node of the tree corresponds to a feature or attribute, and each leaf node represents a class label or a numeric value. The decision tree algorithm makes predictions by traversing the tree from the root node to a leaf node based on the values of the input features. At each internal node, a decision is made based on the value of a specific feature, determining which path to follow. This process continues until a leaf node is reached, which provides the predicted class label.

The algorithm make use of gini index and entropy as criteria to decide which feature to choose in splitting nodes

- Gini index is calculated using the following formula:

$$Gini(D) = 1 - \sum_{i=1}^{m} \mathrm{P}_{\mathrm{i}}^{2}$$

Where P_i is the probability of class i.

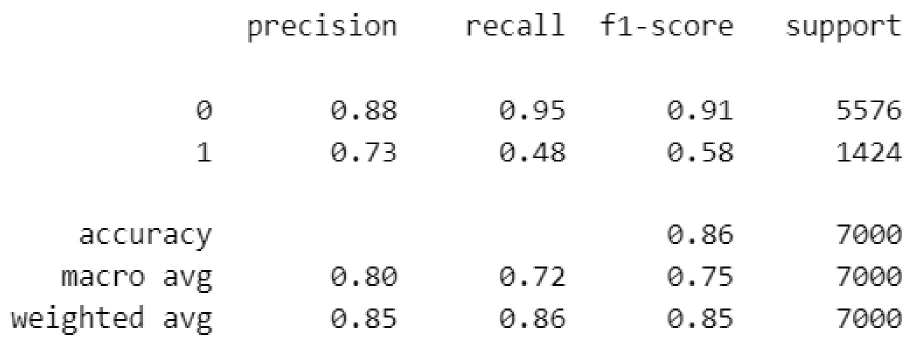
The minimum value of the Gini Index is 0. This happens when the node is pure, this means that all the contained elements in the node are of one unique class. Therefore, this node will not be split again. Thus, the optimum split is chosen by the features with less Gini Index. Moreover, it gets the maximum value when the probabilities of the two classes are the same.

- The entropy is calculated using the following formula:

$$Entropy = -\sum_{i} p_i \times \log_2 p_i$$

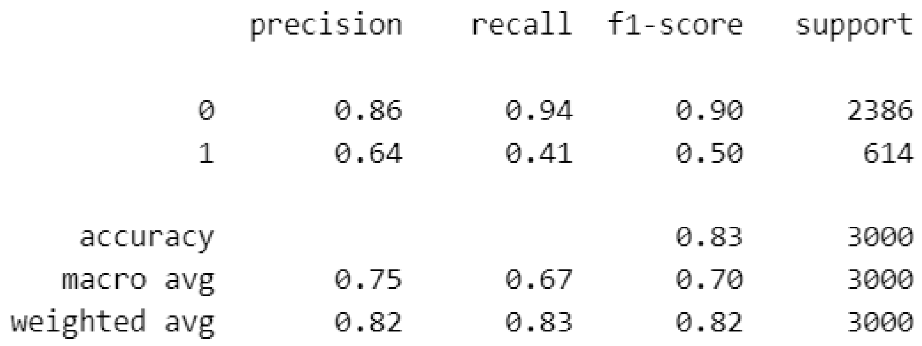
Where P_i is the probability of class i.

Entropy is a measure of information that indicates the disorder of the features with the target. Similar to the Gini Index, the optimum split is chosen by the feature with less entropy. It gets its maximum value when the probability of the two classes is the same and a node is pure when the entropy has its minimum value, which is 0



```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      5576
           1       0.73      0.48      0.58      1424

    accuracy                           0.86      7000
   macro avg       0.80      0.72      0.75      7000
weighted avg       0.85      0.86      0.85      7000
```

Training results



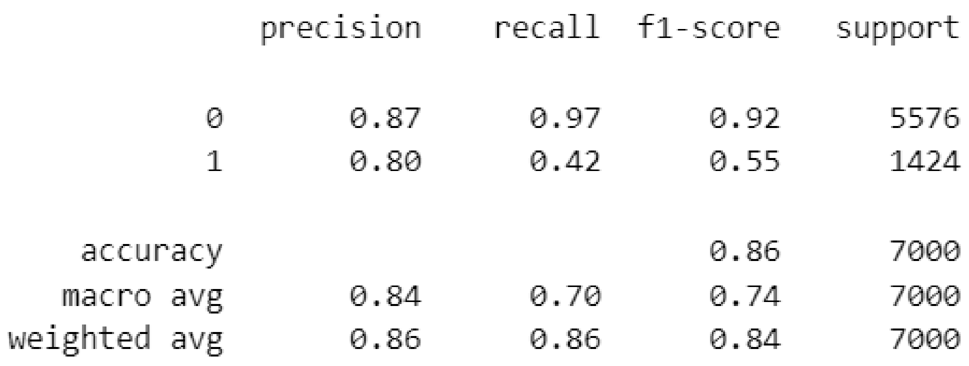
```
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      2386
           1       0.64      0.41      0.50       614

    accuracy                           0.83      3000
   macro avg       0.75      0.67      0.70      3000
weighted avg       0.82      0.83      0.82      3000
```

Testing results

5. Random Forest

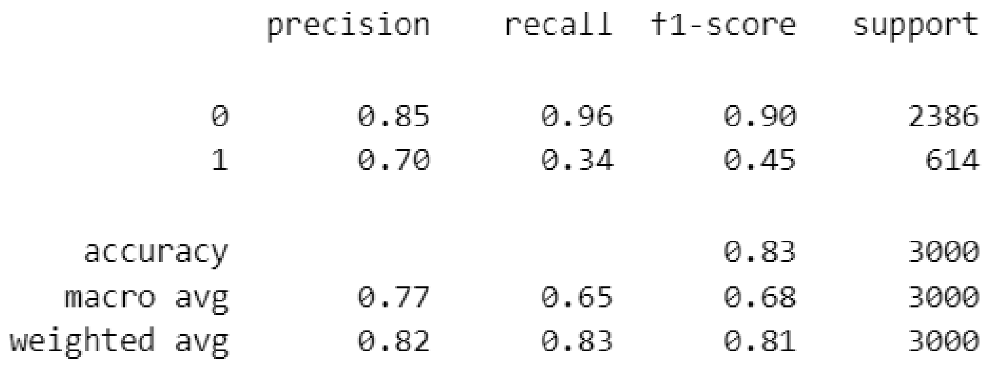
Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions by averaging or voting, resulting in improved accuracy and reduced overfitting. It makes use of bootstrapting (sampling with replacement) and at each node, Random Forest introduces an additional level of randomness by considering only a random subset of the original features.

Ensemble algorithms: combine multiple individual models (usually weak learners such as decision tree) to improve prediction accuracy and robustness.



```
              precision    recall  f1-score   support

           0       0.87      0.97      0.92      5576
           1       0.80      0.42      0.55      1424

    accuracy                           0.86      7000
   macro avg       0.84      0.70      0.74      7000
weighted avg       0.86      0.86      0.84      7000
```

Training results

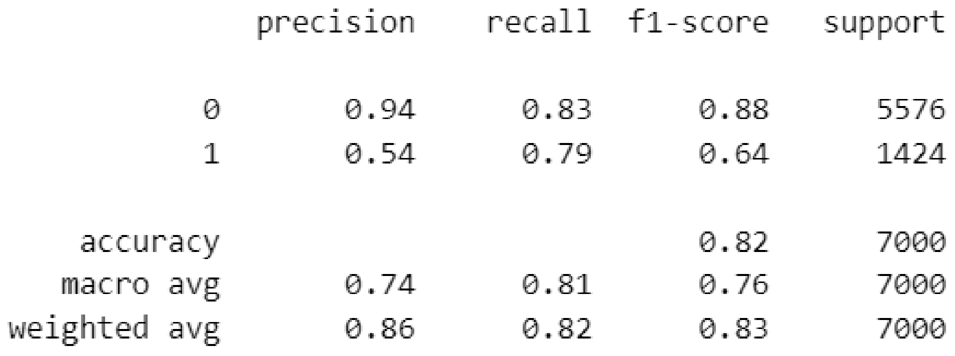


```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      2386
           1       0.70      0.34      0.45       614

    accuracy                           0.83      3000
   macro avg       0.77      0.65      0.68      3000
weighted avg       0.82      0.83      0.81      3000
```

Testing results

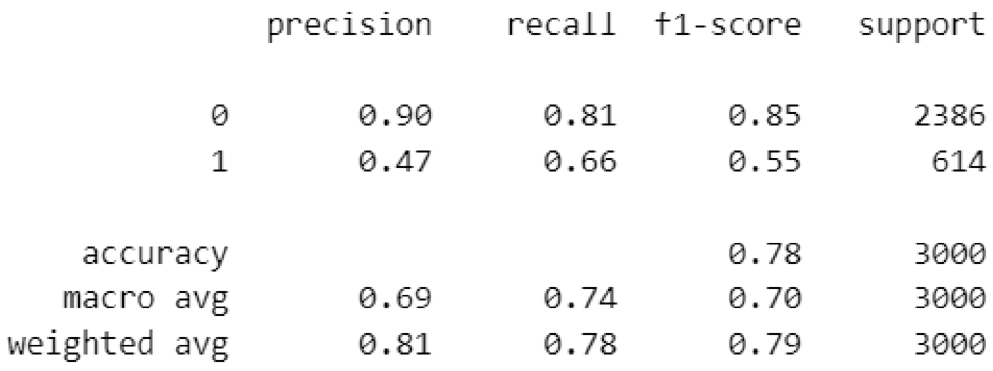
6. XGBoost

XGBoost is an open-source implementation of boosted trees using gradient boosting, an ensemble algorithm that sequentially adds models to correct previous mistakes. It incorporates boosting and regularization methods to optimize performance, especially for regression and classification tasks, exhibiting exceptional speed, scalability, and accuracy on large and intricate datasets.



```
              precision    recall  f1-score   support

           0       0.94      0.83      0.88      5576
           1       0.54      0.79      0.64      1424

    accuracy                           0.82      7000
   macro avg       0.74      0.81      0.76      7000
weighted avg       0.86      0.82      0.83      7000
```

Training results



```
              precision    recall  f1-score   support

           0       0.90      0.81      0.85      2386
           1       0.47      0.66      0.55       614

    accuracy                           0.78      3000
   macro avg       0.69      0.74      0.70      3000
weighted avg       0.81      0.78      0.79      3000
```

Testing results

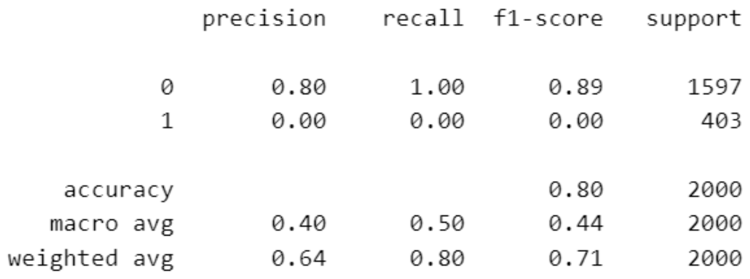
7. Bayesian model:

7.1. Naive Bayes:

We first tried with the assumption that all attributes are independent from each other. The Bayes' theorem simplifies to be as follows:

$$P(exited|gender, \ldots, CreditScore) = \frac{P(gender|exited)\ldots P(CreditScore|exited) * P(exited)}{Z}$$

Here, Z being the normalizing constant.



```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1597
           1       0.00      0.00      0.00       403

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.64      0.80      0.71      2000
```

Testing results

The model seems to have been overfitted. All predictions for the test set are 0, meaning no customers have churned. The assumption that all attributes are independent seems to be too strong.

7.2. Bayesian network:

Since all attributes are independent is rather strong of an assumption for our problem, we decided to use the Bayesian network with the pgmpy library.

We first find a model using HillClimbSearch with K2Score being the loss function.

$$K2(\text{structure}) = \prod_{i=1}^{n} \frac{(\text{data}_i + r - 1)!}{(\text{data}_i - 1)!} \cdot \prod_{j=1}^{m} \frac{(\text{parents}_j + q \cdot r - 1)!}{(\text{parents}_j - 1)!}$$

Where:

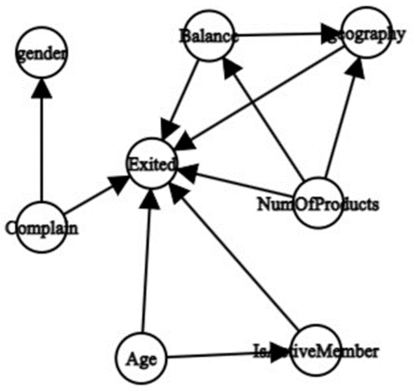
- $K2(\text{structure})$ represents the score of the given network structure.
- n is the number of variables in the network.
- m is the number of variables with parents in the network.
- data_i represents the number of values that the variable i can take.
- parents_j represents the number of possible parent configurations for the variable j.
- r and q are smoothing parameters (usually set to 1).

The result:

[('Balance', 'NumOfProducts'),
('NumOfProducts', 'Exited'),
('IsActiveMember', 'Age'),
('Exited', 'Complain'),
('Exited', 'Age'),
('Exited', 'IsActiveMember'),
('Complain', 'gender'),
('geography', 'Balance'),
('geography', 'Exited')]

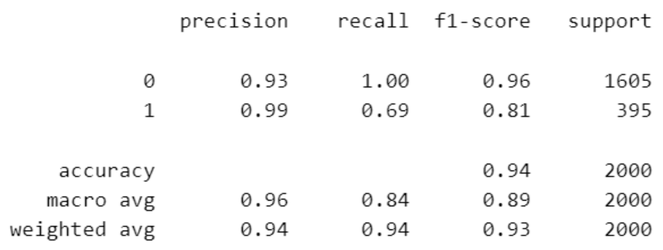
List of edges in the graph

There are a few edges that should be reversed. And there are some vertices that we thought were related turned out to be not.



Resulting Bayesian network

Here are the obtained results:


```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      1605
           1       0.99      0.69      0.81       395

    accuracy                           0.94      2000
   macro avg       0.96      0.84      0.89      2000
weighted avg       0.94      0.94      0.93      2000
```

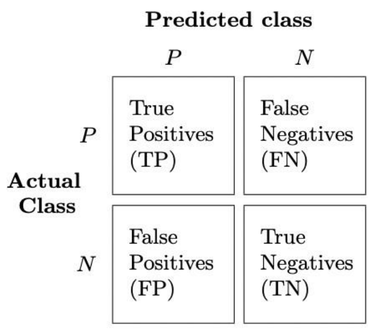
Testing results

The results here seem to be better.

IV. Experimental

1. Performance Metrics

Confusion Matrix. The confusion matrix (or error matrix) is one way to summarize the performance of a classifier for binary classification tasks. This square matrix consists of columns and rows that list the number of instances as absolute or relative ”actual class” and ”predicted class” ratios.



		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Figure 2: Example of representing a confusion matrix.

Accuracy. Both the prediction *error (ERR)* and *accuracy (ACC)* provide general information about how many samples are misclassified. The error can be understood as the sum of all false predictions divided by the number of total predications, and the the accuracy is calculated as the sum of correct predictions divided by the total number of predictions, respectively.

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

Precision, Recall, F1-Score. *Precision* and *Recall* are metrics that are more commonly used in Information Technology and related to the False and True Prositive Rates. In fact, Recall is synonymous to the True Positive Rate and also sometimes called Sensitivity. The *F1-Score* can be understood as a combination of both Precision and Recall.

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{FN + TP}$$

$$F_1 = 2 \times \frac{precision \cdot recall}{precision + recall}$$

Data Splitting. The problem of appropriate data splititng can be handled as a statistical sampling problem. Data simply divided into sets: train, validation and test set.(size of train and validation sets contain 8000 data and test set contains 2000 data)

Cross-validation technique. Cross-validation techniques belong to conventional approaches used to ensure good generalization and to avoid over-training. The basic idea is to divide the dataset T into two subsets – one subset is used for training while the other subset is left out and the performance of the final model is evaluated on it. The main purpose of cross-validation is to achieve a stable and confident estimate of the model performance. Cross-validation techniques can also be used when evaluating and mutually comparing more models, various training algorithms, or when seeking for optimal model parameters.

K-fold cross-validation. The k-fold cross-validation uses a combination of more tests to gain a stable estimate of the model error. The dataset T is divided into k parts of the same size. One parts forms the validation set T_v, the other parts form the training set T_{tr}. This process is repeated for each part of the data.

Algorithm 1 K-fold cross-validation

```
1. Input: dataset $T$, number of folds $k$, performance function $error$, computational models $L_1, \dots, L_m, m \geq 1$
2. Divide $T$ into $k$ disjoint subsets $T_1, \dots, T_k$ of the same size.
3. For $i = 1, \dots, k$:
    $T_v \leftarrow T_i, T_{tr} \leftarrow \{T/T_i\}$
    3.1. For $j = 1, \dots, m$:
        Train model $L_j$ on $T_{tr}$ and periodically use $T_v$ to assess the model performance
        $E_v^j(i) = error(L_j(T_v))$
        Stop training, when a stop-criterion based on $E_v^j(i)$ is satisfied.
4. For $j = 1, \dots, m$, evaluate the performance of the models by: $E_v^j = \frac{1}{k}\sum_{i=1}^{k} E_v^j(i)$ =0
```

In practice, we divided a initial training dataset into 5 folds. One-fifth of those is used as a validation dataset and the remaining folds is used for training. The training process is repeated in 5 times and we estimated the average accuracy in validation data. Then we chose optimal learning parameters from the best accuracy in validation data to examine in testing data.

2. Results

Perfomances on testing data	Accuracy	Precision	Recall	F1-Score
Bayesian network	**0.94**	**0.96**	**0.84**	**0.89**
SVM RBF	0.86	0.81	0.41	0.54
k-Nearest Neighbors	0.73	0.63	0.67	0.64
Decision Tree	0.85	0.79	0.71	0.74
Random Forest	0.86	0.82	0.69	0.73
XGBoost	0.82	0.72	0.75	0.73

Table: Performances of different classifier model

We have tested all proposed machine learning algorithms on the original dataset and Table shows their best performances. The curn prediction using Bayesian network displays highest accuracy of 94%, while KNN classifier has the lowest accuracy of 73%. In terms of the time needed for training, the performances of the machine learning algorithms are similar, with exception of support vector machine (SVM), which is several times slower than the rest, because of the curse of dimension as it increases data dimensions to be able to work with SVM kernel.

V. Conclusion

The finance industry in recent years is a subject of major changes and from a fast-growing industry has come to a state of saturation accompanied with strong competitive market. Customers starve for better services and prices, while their requirements are extremely complex and difficult to understand. In order to cope with this problem, research in this area has the following objectives: finding the influential factors for the customer churn, as well as building a classifier for predicting the customer churn.

In our project, we have explained the methodology for building a classifier model that will predict the customer churn from the data from a fixed telephony operator. We have carefully extracted the customers' behavior within a one-year period, resulting in dataset containing 7043 customers. The results from this study show that predicting the customer churn can be successful with similarly high accuracy. The classification model derived from Bayesian Network have highest performances in recall, F1-Score and ROC in proposed machine learning models.

For future work, we intend to examine some machine learning models based on essemble learning such as LightGBM to improve performances on testing data and research more to understand domain of business for featuring engineering.

References

Telco customer churn - IBM
Available from: Telco customer churn (11.1.3+) - IBM
Telco Customer Churn | Kaggle
Available from: Telco Customer Churn | Kaggle
scikit-learn Package. Available from: seaborn: statistical data visualization — seaborn 0.12.2 documentation