# KNN Classification on Banking Dataset- Marketing Targets (August 2023)

**Anuj Rayamajhi[1], UG, Aayush Regmi[1], UG.**
[1]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

**ABSTRACT** In this study, we explore the application of the K Nearest Neighbors Classifier to predict whether the customer if they will convert or not based on historical banking data of customer. The KNN classifier algorithm, known for its simplicity and efficiency, is used to classify customers into different categories, aiding in informed decision-making and planning. The dataset consists of 16 features having categorical and numeric values. Each data point is labeled with the corresponding features like age, marital status, education, balance, loan, job, default credit, etc. These categories are used as class labels for training the KNN Classifier.

**KEYWORDS** KNN classifier, K-Nearest Neighbors, Banking dataset, Feature selection, etc.

## I. INTRODUCTION

The dataset used pertains to the direct marketing campaigns of a Portuguese banking institution. In these campaigns, the primary objective is to convince clients to subscribe to a term deposit offered by the bank. A term deposit is a financial product where clients invest a sum of money for a fixed period at an agreed-upon interest rate. These deposits are a significant source of income for the bank. To promote term deposits, the bank employs various outreach strategies such as email marketing, advertisements, telephonic marketing, and digital marketing.

While all these strategies have their merits, telephonic marketing campaigns have been proven to be particularly effective in directly engaging with potential clients. However, it's important to note that executing these campaigns requires substantial investments due to the need for large call centers and personnel. Therefore, it becomes crucial to identify those customers who are most likely to subscribe to a term deposit before initiating the campaign. This way, the bank can focus its resources and efforts on targeting these high-potential clients via telephone calls, optimizing the conversion rate and minimizing costs.

The dataset contains a comprehensive set of attributes related to the bank's clients and their interactions with the marketing campaigns. These attributes include demographic information (such as age, job, marital status, education), financial indicators (like average yearly balance, credit default, housing and personal loans), details about the last contact made during the campaign (contact communication type, day, month, duration), and historical data about previous marketing interactions (number of contacts, days since last contact, outcome of previous campaigns). The ultimate goal of this dataset is to predict whether a client will subscribe to a term deposit ("yes" or "no").

In this context, K-Nearest Neighbors (KNN) classification algorithm is going to be used for prediction. KNN is a simple yet effective classification technique. It works by classifying a new data point based on the classes of its nearest neighbors in the feature space. In this dataset, KNN will leverage the various attributes provided to predict whether a client is likely to subscribe to a term deposit or not. By using the KNN classifier, the bank aims to make more informed decisions about which clients to prioritize during their telephonic marketing campaigns, ultimately leading to higher conversion rates and better utilization of resources.

## II. METHODOLOGY

### *A. THEORY*

K-Nearest Neighbors (KNN) is a supervised classification algorithm that operates on the principle of similarity. It assumes that data points with similar features tend to belong to the same class. The fundamental idea is to classify a new data point based on the class labels of its nearest neighbors in the feature space.

### B. MATHEMATICAL EXPLANATION

The mathematics behind KNN Classifier could be broken into multiple steps as:

#### 1) DATA REPRESENTATION

Let's denote our training dataset as

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

Where, $x_i$ represents the feature vector of the $i^{th}$ data point and $y_i$ is its corresponding class label. Each $x_i$ is a $d$ dimensional vector.

$$x_i = [x_{i1}, x_{i2}, \dots, x_{id}]$$

#### 2) CHOOSING K

It is required to select the number of neighbors, *K*, which determines how many nearby data will influence the classification of a new point.

#### 3) DISTANCE METRIC

KNN relies on a distance metric to measure the similarity between data points. The most common distance metrics are Euclidean distance and Manhattan distance.

#### 4) NEAREST NEIGHBORS

For a new, unlabeled data point $x_{new}$, the next step is to find the *K* training data points that are closest to $x_{new}$ based on the chosen distance metric.

#### 5) CLASSIFICATION

Once the *K* nearest neighbors are identified, KNN uses a majority vote to determine the class label of $x_{new}$. That is, among the *K* neighbors, the class label that appears the most frequent is assigned to $x_{new}$.

### C. SYSTEM BLOCK DIAGRAM

The system block diagram consists of multiple blocks which is responsible for their respective functionalities and which makes comprehending the process easier.

At first the dataset is taken from the source. After checking whether the dataset is suitable for classification, the data is checked to see if there are any present outliers, noise or any duplication.

Then dataset cleaning is performed which is crucial as cleaner and comprehensive data leads to more better model performance.

Then the dataset is separated into train and test dataset, and the KNN Classifier is initiated and the train dataset is fitted in the model. The KNN model remembers the whole training dataset and tries to infer the predictions from that.

Then the model is provided with test dataset and train dataset for evaluation and from classification report we could obtain various insights.

### ADVANTAGES OF NAIVE BAYES

KNN has several advantages that make it a popular choice for many applications:

- SIMPLICITY AND EASE OF IMPLEMENTATION: KNN is a straightforward to understand and implement. It doesn't require complex mathematical concepts or parameter tuning.
- NO TRAINING PHASE: Unlike many other machine learning algorithms, KNN doesn't involve a training phase. It simply stores the training data and uses it for making predictions.
- FLEXIBILITY IN DATA: KNN can handle both classification and regression tasks, making it versatile for a variety of problem types.
- INTERPRETABLE RESULTS: KNN's decision-making process is easy to interpret. Predictions are based on the class labels of the nearest neighbors.
- NON-PARAMETRIC: KNN is a non-parametric algorithm, meaning it makes no assumptions about the underlying data distribution. This flexibility makes it suitable for various types of data.

### LIMITATIONS OF NAIVE BAYES

While KNN is effective in many cases, it does have some limitations:

- COMPUTATIONAL COMPLEXITY: As the dataset grows, the computational cost of KNN increases significantly. Calculating distances to all training points can become time-consuming.
- CURSE OF DIMENSIONALITY: KNN's performance can degrade in high-dimensional spaces. As the number of dimensions increases, data points become equally spaced, leading to decreased discrimination between neighbors.
- NEED FOR PROPER SCALING: KNN is sensitive to the scale of features. Features with larger scales can dominate the distance calculations. It's important to normalize or standardize features.
- IMBALANCED DATA: In datasets with imbalanced class distributions, KNN can be biased toward the majority class due to the voting mechanism.
- LOCAL OPTIMA: KNN might get stuck in local optima when there are varying densities of data points in different regions of the feature space.

## III. Exploratory Data Analysis

The 'Banking Dataset-Marketing Targets' dataset contains 16 feature columns and 1 target column (yes or no).

TABLE I
FIRST 5 ENTRIES OF DATASET

| age | Job | Marital | education | default | balance | Housing |
|---|---|---|---|---|---|---|
| 58 | Management | married | tertiary | no | 2143 | yes |
| 44 | technician | single | secondary | no | 29 | yes |
| 33 | Entrepreneur | married | secondary | no | 2 | yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| 47 | Blue-collar | married | unknown | no | 1506 | yes |
| 33 | unknown | single | unknown | no | 1 | no |

Above snapshot table is a portion of dataset, containing 7 features, but the actual dataset contains 16 features which is not possible to be included in the report. The dataset contains demographic information (such as age, job, marital status, education), financial indicators (like average yearly balance, credit default, housing and personal loans), details about the last contact made during the campaign (contact communication type, day, month, duration), and historical data about previous marketing interactions (number of contacts, days since last contact, outcome of previous campaigns). The ultimate goal of this dataset is to predict whether a client will subscribe to a term deposit ("yes" or "no").

The dataset had 4 attributes with numeric values, namely: age, balance, campaign, and previous. The analysis of these features is mentioned in figure 1. Figure 2-5 depicts the histogram and density plot of these attributes.

Most features also have categorical values:

- **Job:** This attribute encapsulates the diverse spectrum of professions that clients belong to. Ranging from positions like "admin." and "management" to "blue-collar" and more, it provides a glimpse into the occupational diversity within the client base.
- **Marital Status:** Capturing the relational dimension, the "marital status" attribute categorizes individuals into groups such as "married," "divorced," or "single." Importantly, it amalgamates both those who are divorced and those who are widowed under the "divorced" category.
- **Education**: Representing the educational attainment of clients, the "education" feature classifies them based on levels like "unknown," "secondary," "primary," and "tertiary." This paints a picture of the clients' educational backgrounds, potentially influencing their financial decisions.
- **Default:** With a binary nature, the "default" attribute signifies whether clients have credit in default or not. This attribute assumes the values of "yes" or "no," and can provide insights into clients' financial stability.
- **Housing and Loan:** These binary attributes— "housing" and "loan"—offer insight into the financial obligations of clients. "Housing" indicates whether clients have housing loans, while "loan" reveals the presence of personal loans. Their binary nature, with values "yes" or "no," highlights key financial aspects.
- **Contact and Month**: The "contact" attribute delves into communication methods used during the last campaign, featuring categories such as "unknown," "telephone," and "cellular." Meanwhile, "month" captures the last contact month using abbreviated forms like "jan," "feb," etc. These attributes detail the campaign interaction dynamics and timing.
- **Poutcome**: A categorical feature detailing the outcome of prior marketing efforts, "poutcome" encompasses categories like "unknown," "other," "failure," and "success." This offers insights into the historical effectiveness of marketing endeavors.

## IV. Results:

The classification report provides a comprehensive evaluation of the performance of the K-Nearest Neighbors (KNN) classifier on both the training and test sets. The report includes metrics such as precision, recall, and F1-score, which offer insights into the classifier's ability to correctly identify positive cases ("yes") and negative cases ("no"). Additionally, accuracy, which represents the overall correctness of the predictions, is also presented.

Precision, recall, and F1-score are key metrics for evaluating a classifier's performance, particularly in imbalanced datasets where one class is much more prevalent than the other.

- **Precision:** This metric indicates the proportion of correctly predicted positive cases among all instances that were predicted as positive. In other words, it quantifies how well the classifier avoids false positives. A higher precision indicates that the classifier is making fewer incorrect positive predictions.
- **Recall (Sensitivity or True Positive Rate):** Recall measures the proportion of correctly predicted positive cases out of all actual positive cases. It indicates the classifier's ability to capture positive cases and avoid false negatives. A higher recall suggests that the classifier is effectively identifying positive cases.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall. It provides a balanced measure that considers both false positives and false negatives. F1-score is particularly useful when dealing with imbalanced datasets, as it combines the strengths of precision and recall.

Looking at the training set results:

- The accuracy of approximately 90% indicates that the classifier is performing well on this set.
- The precision for class '0' (no) is high (0.91), indicating that the classifier is accurate in identifying instances of 'no'.
- However, the precision for class '1' (yes) is relatively low (0.71), implying that there are false positives for 'yes' predictions.
- The recall for class '0' is high (0.99), indicating that the classifier is capturing the majority of instances for 'no'.
- The recall for class '1' is low (0.27), suggesting that the classifier is not effectively identifying instances of 'yes'.
- The F1-score for class '0' is high (0.95), showing a good balance between precision and recall.

- The F1-score for class '1' is relatively low (0.40), reflecting the challenge of correctly predicting instances of 'yes' due to their low prevalence.

Analyzing the test set results:

- The accuracy of approximately 88% indicates that the classifier's performance is consistent with the training set.
- The precision for class '0' remains high (0.90), indicating accurate 'no' predictions.
- However, the precision for class '1' is lower (0.44), indicating a tendency for false positives in 'yes' predictions.
- The recall for class '0' is high (0.97), confirming the classifier's ability to capture instances of 'no'.
- The recall for class '1' remains low (0.18), highlighting the difficulty in identifying instances of 'yes'.
- The F1-score for class '0' is relatively high (0.93), representing a good balance between precision and recall for 'no'.
- The F1-score for class '1' remains low (0.25), indicating challenges in achieving a balance between precision and recall for 'yes'.

These results were obtained when the KNN classifier was fitted in the whole dataset without any feature selection. The evaluation of the K-Nearest Neighbors (KNN) classifier after feature selection using correlation is crucial for understanding how this preprocessing step has impacted the classifier's performance. Feature selection, in this case, refers to the process of choosing a subset of relevant features based on their correlation with the target variable. Analysis on the new classification report obtained after applying this technique to both the training and test sets:

Comparison of Train Set Results:

- **Accuracy**: The accuracy remains consistent at approximately 88%, indicating that the overall performance of the classifier is not substantially altered.
- **Class '0' (no) Predictions**: The precision for class '0' remains high (0.88), signifying accurate predictions of 'no' instances.
- **Class '1' (yes) Predictions**: However, there's a significant change in the precision and recall for class '1'. The precision drops to 0.70, which indicates that the classifier is less accurate in predicting 'yes' instances. Additionally, the recall for class '1' remains extremely low at 0.02, which suggests that the classifier captures very few 'yes' instances.
- **F1-Scores**: The F1-score for class '0' remains relatively high (0.94), maintaining a good balance between precision and recall. However, the F1-score for class '1' drops to 0.04, indicating challenges in achieving a balance between precision and recall for 'yes'.

Comparison of Test Set Results:

- **Accuracy:** Similar to the training set, the accuracy remains stable at approximately 89%.
- **Class '0' (no) Predictions:** The precision for class '0' stays high (0.89), ensuring accurate predictions for 'no' instances.
- **Class '1' (yes) Predictions:** For class '1', the precision improves to 0.78, suggesting better accuracy in 'yes' predictions. However, the recall remains low at 0.03, indicating that the classifier still struggles to capture 'yes' instances effectively.
- **F1-Scores:** The F1-score for class '0' maintains its high value (0.94), while the F1-score for class '1' improves slightly to 0.06. This demonstrates some progress in achieving a balance between precision and recall for 'yes'.

The shift in precision and recall for class '1' after feature selection suggests that this technique had a notable impact on the classifier's ability to predict 'yes' instances. While precision improved, recall remained low, indicating that although the classifier is making fewer false positive 'yes' predictions, it is still struggling to identify actual 'yes' instances.

## V. DISCUSSION AND ANALYSIS

Given the high prevalence of 'no' cases relative to 'yes', the KNN classifier's challenges in accurately predicting 'yes' instances are understandable. The lower recall and F1-score for class '1' are indicative of this imbalance. To improve the classifier's performance on 'yes' predictions:

- **Address Imbalance**: Strategies like oversampling the minority class ('yes') or using different evaluation metrics (e.g., area under the precision-recall curve) could provide a more nuanced view of performance.
- **Consider Model Selection:** Experiment with other classification algorithms that might handle imbalanced data better, such as ensemble methods like Random Forest or boosting techniques.
- **Hyperparameter Tuning:** Experiment with different values of 'k' and explore other distance metrics to optimize the model's performance.

### *Imbalance and Its Impact*

The dataset's class distribution, with a significant majority of 'no' instances and relatively few 'yes' instances, creates a skewed learning environment. The classifier becomes biased toward the majority class, resulting in a higher emphasis on predicting 'no' instances accurately. Consequently, the classifier struggles to learn the patterns and characteristics of the minority class, leading to lower recall and F1-scores for 'yes'.

Strategies to Address Imbalance:

- **Resampling Techniques**: Oversampling the minority class or under sampling the majority class can help balance the class distribution, providing the classifier with a more equal representation of both classes during training. Techniques like Synthetic Minority Over-

sampling Technique (SMOTE) can be particularly effective in generating synthetic instances of the minority class.

- **Evaluation Metrics:** Due to the class imbalance, it's important to focus on evaluation metrics that are sensitive to the minority class, such as precision-recall curves, area under the precision-recall curve, or F1-score. These metrics provide a clearer picture of the classifier's ability to correctly predict positive cases.

### *Exploring Alternative Algorithms:*

KNN is not the only algorithm suitable for this problem. Alternative classification algorithms can be explored to address the challenges posed by class imbalance. Ensemble methods like Random Forest and boosting techniques (e.g., AdaBoost, XGBoost) are known to handle imbalanced data better. These algorithms inherently manage class imbalances and provide mechanisms for assigning different weights to classes during training.

## VI. CONCLUSION

The reported results of the KNN classifier on the training and test sets highlight the challenges of dealing with class imbalance. The classifier's strengths lie in accurately predicting the majority class ('no'), but it struggles with the minority class ('yes'). By implementing resampling techniques, exploring alternative algorithms, tuning hyperparameters, and ensuring proper data preprocessing, it's possible to mitigate the effects of class imbalance and achieve a more balanced and accurate prediction model. Ultimately, a combination of these strategies is likely to yield the best results, enhancing the classifier's performance across both 'yes' and 'no' instances.

The feature selection process based on correlation has influenced the classifier's performance, particularly in relation to 'yes' predictions. Although there's a noticeable improvement in precision, the challenges with low recall persist. It's imperative to carefully analyze the selected features, consider additional preprocessing steps, and explore techniques that address class imbalance to achieve a more balanced and accurate KNN classifier, especially for 'yes' instances.

## VII. REFERENCES

[1] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

**ANUJ RAYAMAJHI** is an ambitious individual currently in the final year of his Bachelor's degree program in Computer Engineering at the esteemed Institute of Engineering Thapathali Campus. With a keen interest in various fields such as Artificial Intelligence, Machine Learning, Data Science, Software Development, and Engineering, Anuj possesses a diverse range of skills and a thirst for knowledge. He constantly seeks out opportunities to enhance his understanding of these subjects through research, online courses, and practical experience.

Apart from his academic pursuits, Anuj has a passion for music, sports, and computer games. In his leisure time, he enjoys exploring different genres of music, engaging in sports activities to stay active, and immersing himself in the virtual worlds of computer games.

With a strong foundation in computer engineering and a multifaceted interest in emerging technologies, Anuj is driven to make significant contributions in the fields of AI, machine learning, and data science. He strives to stay up-to-date with the latest advancements in these domains, continuously expanding his knowledge and honing his skills. Anuj's dedication, enthusiasm, and well-rounded interests make him a promising individual poised to excel in the ever-evolving field of technology

**AAYUSH REGMI** is a dynamic individual currently pursuing his Bachelor's degree in Computer Engineering at the renowned Institute of Engineering Thapathali Campus. With a strong inclination towards technology, Aayush has developed a keen interest in a wide range of fields, including Artificial Intelligence, Machine Learning, Data Science, Web Development, Quantum Computing, and more. His diverse expertise reflects his commitment to exploring various facets of computer science and pushing the boundaries of innovation.

In addition to his academic pursuits, Aayush finds solace and joy in his hobbies. As an avid sports enthusiast, he actively engages in cricket and football, relishing the thrill of competition and teamwork. Aayush also has a creative side and enjoys playing musical instruments, finding harmony in the melodies he creates.

With an ever-curious mind and a passion for learning, Aayush strives to stay ahead in the rapidly evolving world of technology. He is particularly intrigued by the emerging field of Quantum Computing and its potential to revolutionize the computing landscape. Aayush's dedication, coupled with his diverse skill set, positions him to make significant contributions in web development, quantum computing, and other innovative domains.

Driven by a relentless pursuit of knowledge and a desire to create a positive impact, Aayush is determined to shape the future through his expertise in computer engineering. His enthusiasm, adaptability, and holistic interests make him a promising individual poised to excel in the dynamic and ever-expanding field of technology.

## APPENDIX A: FIGURES

| | age | balance | campaign | previous |
|---|---|---|---|---|
| count | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 |
| mean | 40.936210 | 1362.272058 | 2.763841 | 0.580323 |
| std | 10.618762 | 3044.765829 | 3.098021 | 2.303441 |
| min | 18.000000 | -8019.000000 | 1.000000 | 0.000000 |
| 25% | 33.000000 | 72.000000 | 1.000000 | 0.000000 |
| 50% | 39.000000 | 448.000000 | 2.000000 | 0.000000 |
| 75% | 48.000000 | 1428.000000 | 3.000000 | 0.000000 |
| max | 95.000000 | 102127.000000 | 63.000000 | 275.000000 |

**Figure 1 :** Numeric Features Analysis

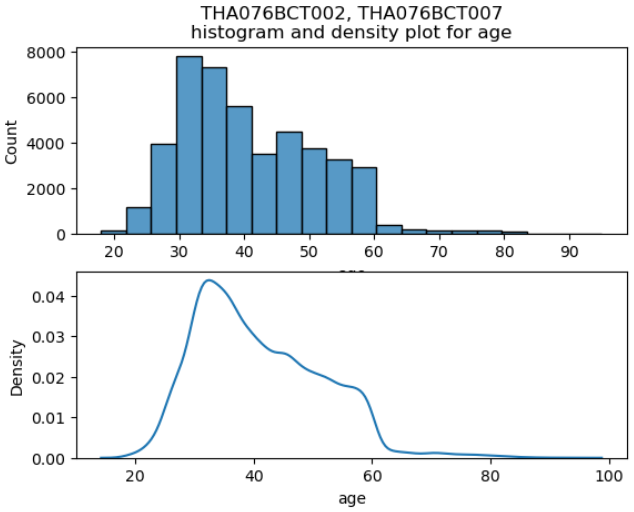


**Figure 2 :** Age Distribution plot

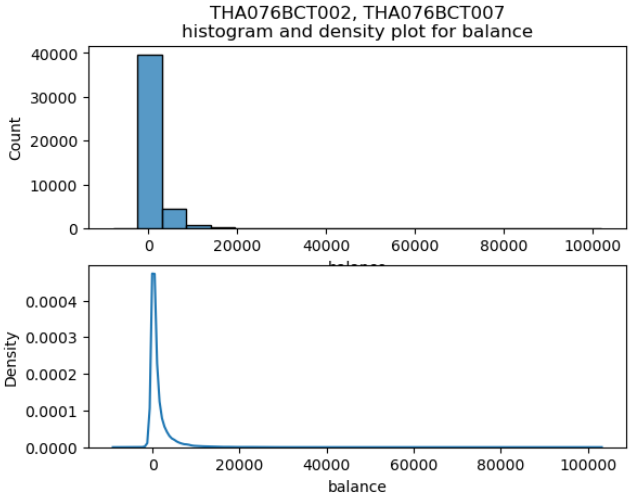


**Figure 3 :** Balance Distribution Plot

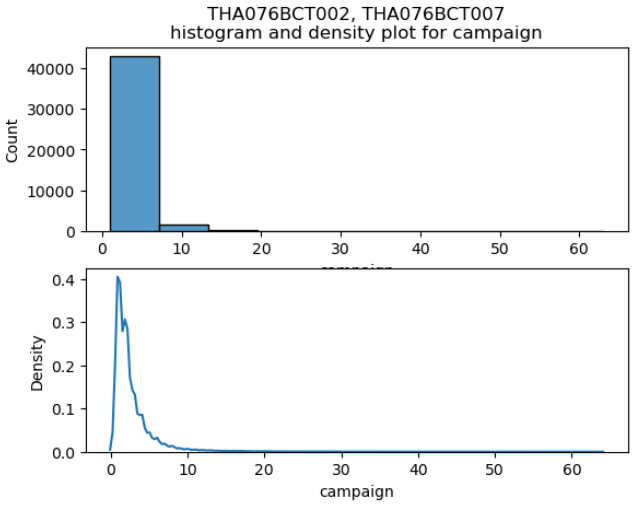


**Figure 4 :** Campaign Distribution Plot

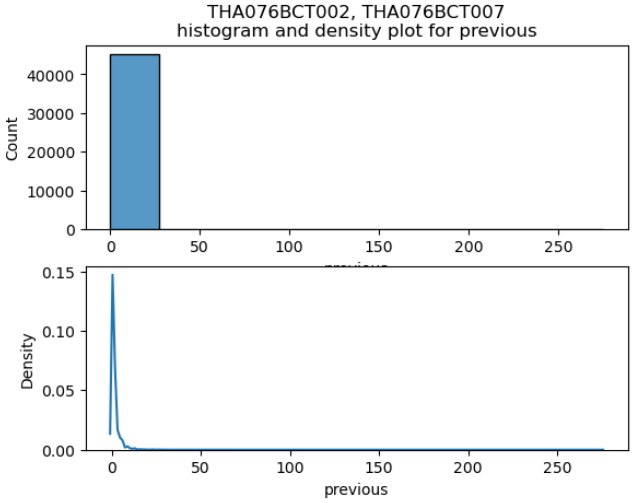


**Figure 5 :** 'Previous' distribution plot

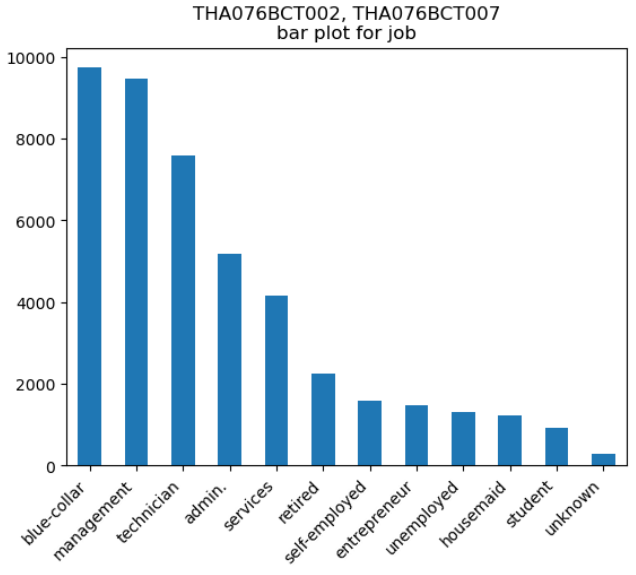


**Figure 6 :** Job Category Bar plot

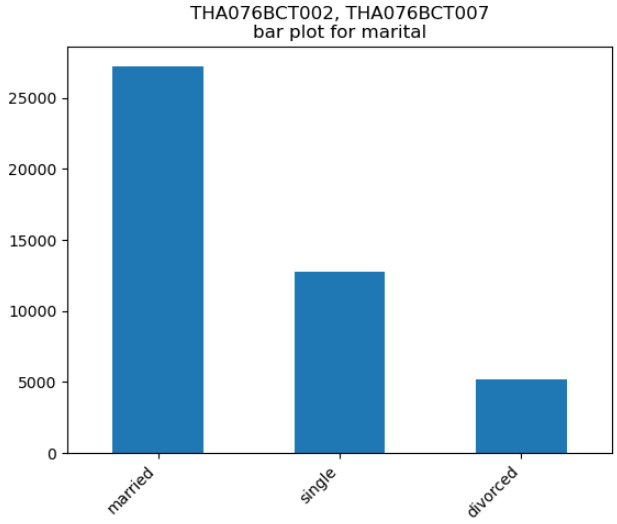


**Figure 7 :** Marital Category Bar Plot

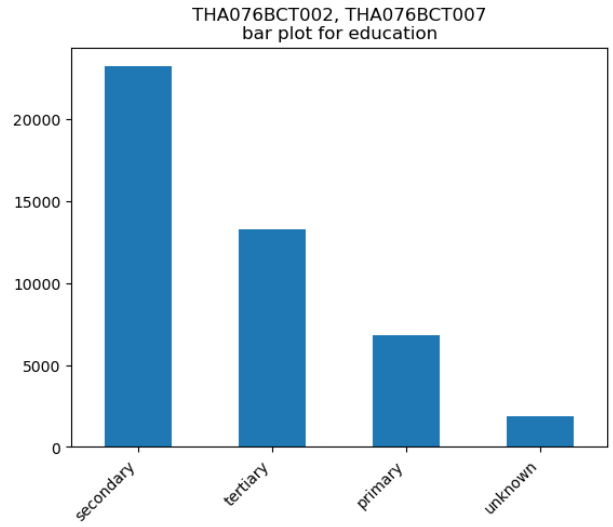


**Figure 8 :** Education Category Bar Plot

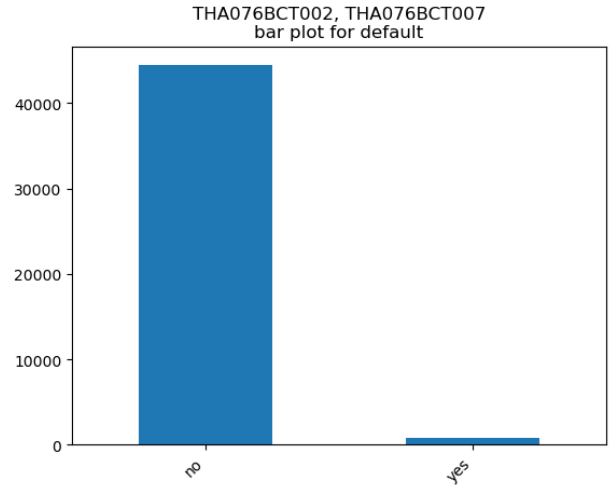


**Figure 9 :** Default Credit Bar plot

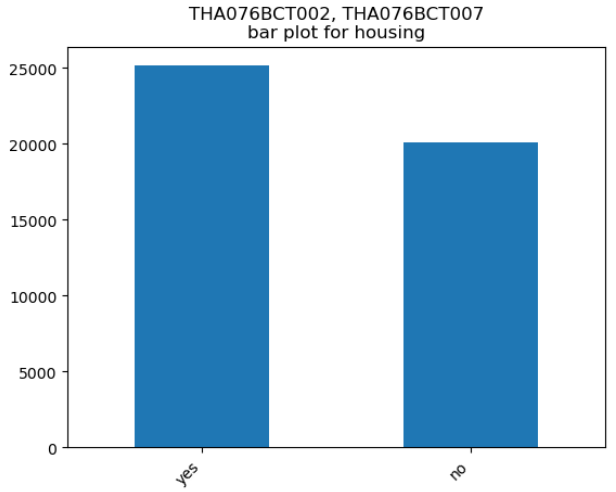


**Figure 10 :** Housing loan Bar Plot

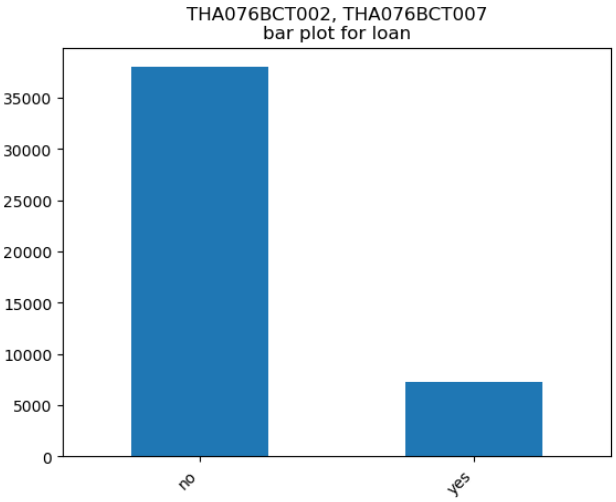


**Figure 11 :** Personal Loan Bar Plot

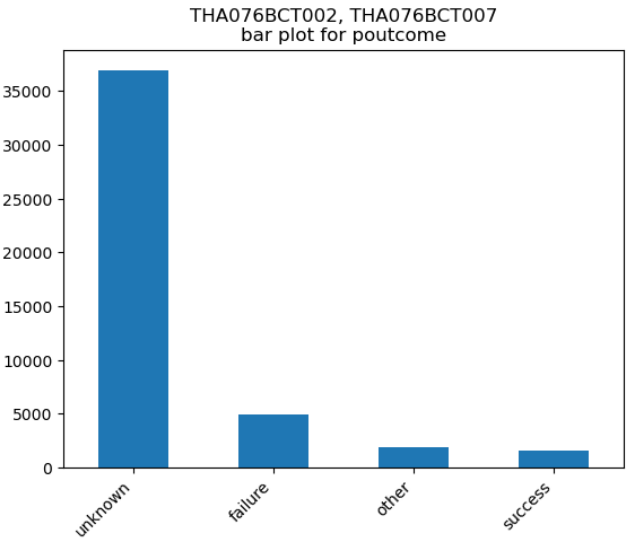


**Figure 12 :** 'poutcome' Bar Plot

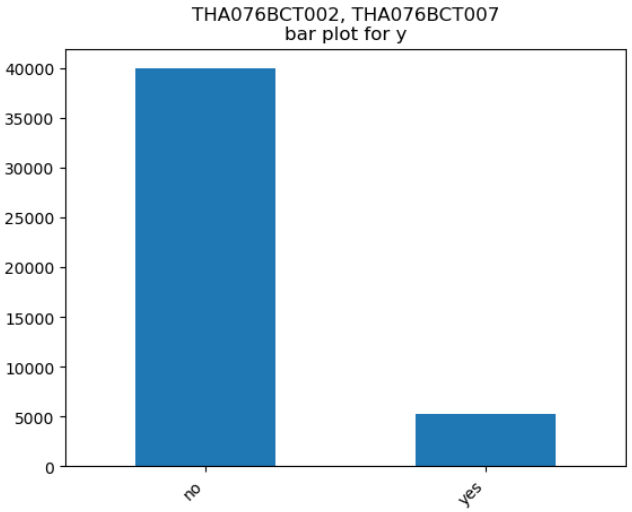


**Figure 13 :** **Target Label Bar Plot**

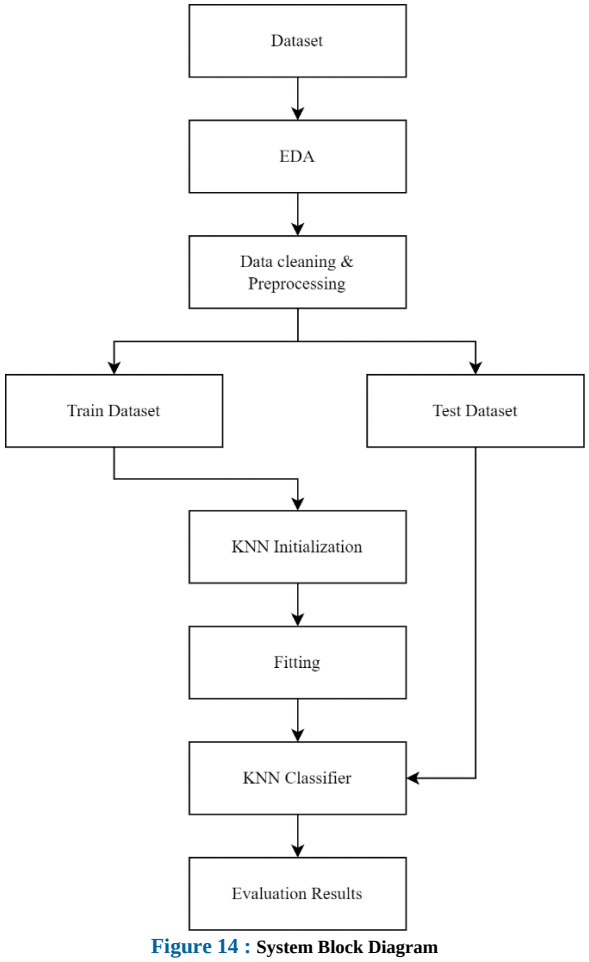


**Figure 14 :** **System Block Diagram**

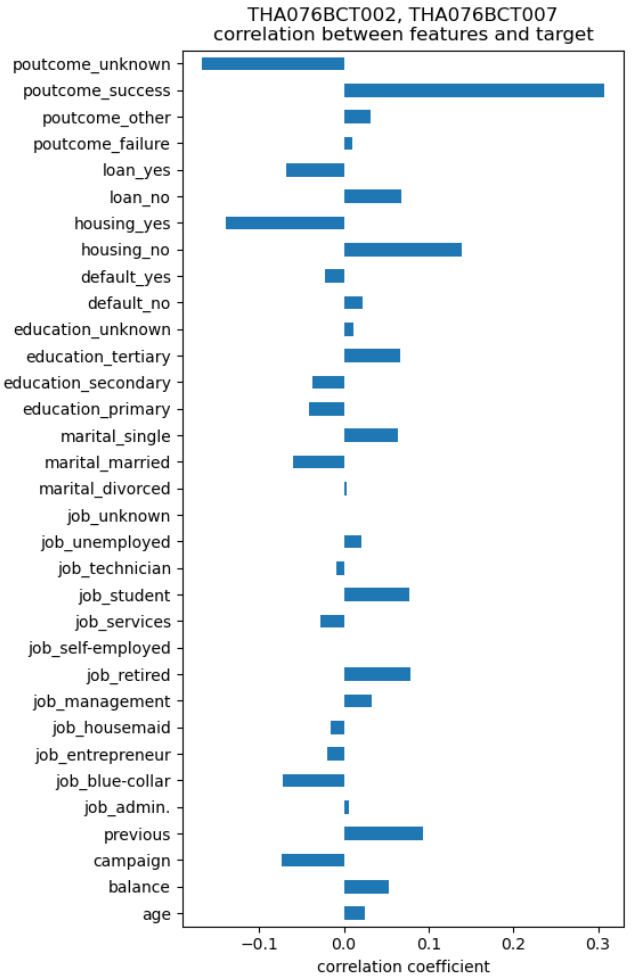


**Figure 15 :** **Correlation between features and target**

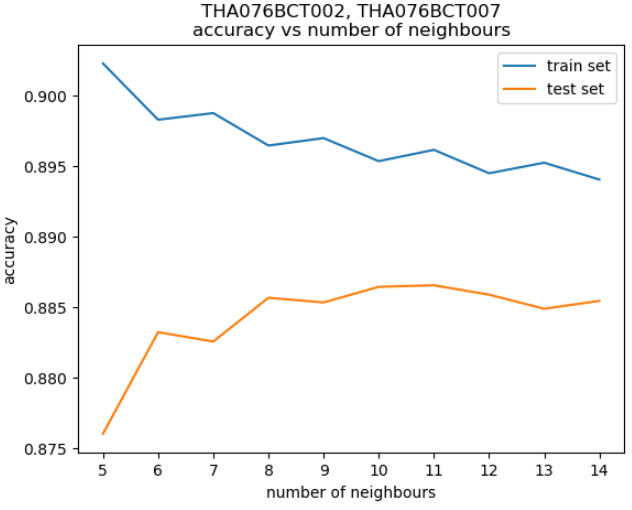


**Figure 16 :** **Accuracy vs K values**

```
For Train Set
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     31958
           1       0.74      0.23      0.35      4210

    accuracy                           0.90     36168
   macro avg       0.82      0.61      0.65     36168
weighted avg       0.89      0.90      0.88     36168

For Test Set
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      7964
           1       0.31      0.09      0.15      1079

    accuracy                           0.87      9043
   macro avg       0.60      0.53      0.54      9043
weighted avg       0.82      0.87      0.83      9043
```

**Figure 18:** Classification report of KNN model (whole data)

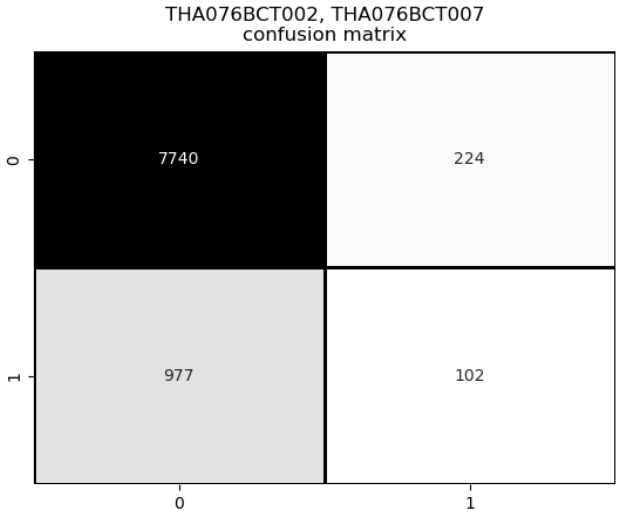


**Figure 17 :** Confusion Matrix of KNN Classifier (whole data)

## APPENDIX B : CODE

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

import warnings

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import cross_val_score,
GridSearchCV

%matplotlib inline



warnings.filterwarnings('ignore')



def transform_to_csv():


    file = open('./train.csv')

    new = []

    for line in file.readlines():

        l = line

        l = l.replace(";", ',')

        new.append(l)
```

```
    file.close()


    file = open("./train_mod.csv", "w")
    file.write(''.join(new))
    file.close()


    file = open('./test.csv')
    new = []
    for line in file.readlines():
        l = line
        l = l.replace(";", ',')
        new.append(l)
    file.close()


    file = open("./test_mod.csv", "w")
    file.write(''.join(new))
    file.close()


transform_to_csv()


data = pd.read_csv('train_mod.csv').drop(['month',
'duration', 'day', 'contact', 'pdays'], axis=1)
data
data.describe()
def plot_column_bar(dataframe, col):
    dataframe[col].value_counts().plot.bar()
    plt.title(f"THA076BCT002, THA076BCT007\nbar plot
for {col}")
    plt.xticks(rotation=45, ha='right')
    plt.show()


plot_column_bar(data, 'job')
plot_column_bar(data, 'marital')
plot_column_bar(data, 'education')
plot_column_bar(data, 'default')
plot_column_bar(data, 'housing')
plot_column_bar(data, 'loan')
plot_column_bar(data, 'poutcome')
plot_column_bar(data, 'y')


## Before normalization
def plot_column(dataframe, col, bins=None):
    plt.subplot(2,1, 1)
    if bins == None:
        sns.hist.plot(dataframe[col])
    else:
        sns.histplot(dataframe[col], bins=bins)


    plt.title(f"THA076BCT002, THA076BCT007\nhistogram
and density plot for {col}")
    plt.subplot(2,1,2)
    sns.kdeplot(data[col])
```

```
    plt.show()

plot_column(data, 'age', bins=20)

plot_column(data, 'balance', bins=20)

plot_column(data, 'campaign', bins=10)

plot_column(data, 'previous', bins=10)

X = data.drop(['y'], axis=1)

X = pd.get_dummies(X)

X.head(2)

y =
pd.get_dummies(data['y'].to_frame())['y_yes'].to_frame
()

y.head()

X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2)

knn_clf = KNeighborsClassifier()

knn_clf.fit(X_train, y_train.to_numpy().ravel())

cm = confusion_matrix(y_test, knn_clf.predict(X_test))

sns.heatmap(cm, linecolor='k', linewidths=1,
cmap='binary', annot=True, fmt="g", cbar=False)

plt.title("THA076BCT002, THA076BCT007\nconfusion
matrix")



print("For Train Set")

print(classification_report(y_train,
knn_clf.predict(X_train)))



print("For Test Set")

print(classification_report(y_test,
knn_clf.predict(X_test)))
```

```
## After Normalization

data[['age', 'balance', 'campaign',
'previous']].head(5)

data_nor =
pd.DataFrame(StandardScaler().fit_transform(data[['age
', 'balance', 'campaign', 'previous']]),
columns=['age', 'balance', 'campaign', 'previous'])

data_nor.describe()

new_data = data.copy()

new_data[['age', 'balance', 'campaign', 'previous']] =
data_nor

new_data.head(2)

X = new_data.drop(['y'], axis=1)

X = pd.get_dummies(X)

X.head(2)

y =
pd.get_dummies(data['y'].to_frame())['y_yes'].to_frame
()

y.head()

X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2)

knn_clf = KNeighborsClassifier()

knn_clf.fit(X_train, y_train.to_numpy().ravel())

cm = confusion_matrix(y_test, knn_clf.predict(X_test))

sns.heatmap(cm, linecolor='k', linewidths=1,
cmap='binary', annot=True, fmt="g", cbar=False)

plt.title("THA076BCT002, THA076BCT007\nconfusion
matrix")



print("For Train Set")
```

```
print(classification_report(y_train,
knn_clf.predict(X_train)))



print("For Test Set")

print(classification_report(y_test,
knn_clf.predict(X_test)))

def get_accuracy(x_train, y_train, x_test, y_test):

    train_accuracy_cache = {}

    test_accuracy_cache = {}



    for i in range(5, 15):

        knn_clf = KNeighborsClassifier(n_neighbors=i)

        knn_clf.fit(x_train,
y_train.to_numpy().ravel())



        train_accuracy_cache[str(i)] =
accuracy_score(y_train, knn_clf.predict(x_train))

        test_accuracy_cache[str(i)] =
accuracy_score(y_test, knn_clf.predict(x_test))



    return train_accuracy_cache, test_accuracy_cache



train_cache, test_cache = get_accuracy(X_train,
y_train, X_test, y_test)

train_cache, test_cache

plt.plot(train_cache.keys(), train_cache.values(),
label="train set")

plt.plot(test_cache.keys(), test_cache.values(),
label="test set")

plt.title("THA076BCT002, THA076BCT007\naccuracy vs
number of neighbours")

plt.xlabel("number of neighbours")

plt.ylabel("accuracy")

plt.legend()

plt.show()

plt.plot(test_cache.keys(), test_cache.values())

plt.title("THA076BCT002, THA076BCT007\naccuracy vs
number of neighbours on train set")

plt.xlabel("number of neighbours")

plt.ylabel("accuracy")

plt.show()

pd.get_dummies(new_data).corr()['y_yes'].drop(['y_yes'
, 'y_no']).sort_values(ascending=True)

## Feature Selection

### Using correlation

plt.figure(figsize=(5,10))

corr =
pd.get_dummies(new_data).corr()['y_yes'].drop(['y_yes'
, 'y_no'])

ax = corr.sort_values(ascending=True,
key=abs).plot.barh()

plt.title("THA076BCT002, THA076BCT007\ncorrelation
between features and target")

plt.xlabel("correlation coefficient")

plt.show()

new_features =
pd.get_dummies(new_data)[['poutcome_success',
'poutcome_unknown', 'housing_no',
```

```
       'housing_yes', 'job_retired', 'y_yes']]

X = new_features.drop(['y_yes'], axis=1)

X.head(2)

y = new_features['y_yes'].to_frame()

y.head()

X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2)

knn_clf = KNeighborsClassifier()

knn_clf.fit(X_train, y_train.to_numpy().ravel())

y_train.sum()

cm = confusion_matrix(y_train,
knn_clf.predict(X_train))

sns.heatmap(cm, linecolor='k', linewidths=1,
cmap='binary', annot=True, fmt="g", cbar=False)

plt.title("THA076BCT002, THA076BCT007\nconfusion
matrix")

plt.show()

cm = confusion_matrix(y_test, knn_clf.predict(X_test))

sns.heatmap(cm, linecolor='k', linewidths=1,
cmap='binary', annot=True, fmt="g", cbar=False)

plt.title("THA076BCT002, THA076BCT007\nconfusion
matrix")

plt.show()

print("For Train Set")

print(classification_report(y_train,
knn_clf.predict(X_train)))



print("For Test Set")

print(classification_report(y_test,
knn_clf.predict(X_test)))

### Using Fisher's score

from skfeature.function.similarity_based import
fisher_score

ranks =
fisher_score.fisher_score(pd.get_dummies(new_data.drop
('y', axis=1)), pd.get_dummies(new_data['y'])['yes'])
```