BANK MARKETING DATA CLASSIFICATION USING NAVIS BAYES CLASSIFIER

# ABSTRACT:

Nowadays, the banking industry is no exception to the general trend of massive data production in all spheres of modern life. In this research, we analyze the categorization of marketing data from banks using a variety of machine learning techniques. The term "banking" refers to the supply of services by a bank to an individual consumer. The data was first compiled from the UCI Machine Learning repository and the Kaggle website. Phone-based banking marketing statistics are the focus of this data set. Python is utilized as the language of implementation, and the Machine Learning concept is employed for statistical learning and data analysis in this work. An improved prediction is the primary goal of machine learning's model-building phase. In order to classify the results, a supervised Naive Bayes algorithm is used to the data. The primary goal of the modeling effort is to characterize whether or not the consumer has chosen a term deposit. The bank should devote substantial time to returning phone calls from prospective customers. Accuracy, precision, recall, and F1 score were all evaluated as a consequence of this study in the direction of term deposit forecasting.

Based on our analysis of the Naive Bayes classifier on the bank marketing data, we can conclude that while the classifier's performance is moderate, there is certainly room for improvement.

Our results show that the classifier achieved an accuracy of 0.681, which indicates that it was able to correctly classify 68.1% of instances. However, its precision of 0.756 and recall of 0.493 indicate that it struggled to correctly identify positive instances, which is a common issue with unbalanced datasets like the one we used.

# INRODUCTION:

Banks were financial institutions that accepted deposits from their customers and provided loans to them in exchange for interest. To better banking tactics and maintain solid client relationships, banks retained vast amounts of information about their consumers. A bank's customers were its most valuable asset. Direct marketing referred to the practice of advertising a product or service by making direct contact with its target audience, be it by traditional mail, electronic mail, human contact, mobile phone, or any other means of communication. Attracting new banking clients was a primary marketing goal [1]. The classification objective was to foretell if a consumer would subscribe the term deposit using data obtained from the UCI machine learning repository[2]. Machine learning techniques for data analysis methods and automating analytical building models to predict the accuracy of bank customer data were used in this work. Python was used as the programming language[12], and its high-level, interpreter and extensive standard library were freely available sources for all major platforms from the Python website. With the use of a classification technique, such as the Naive Bayes classifier algorithm, the most accurate results possible were obtained by measuring how well each instance in a dataset was defined by its properties. The bank should have gone after prospective clients who had spent a lot of time responding to bank calls. The overarching goal of this research was to construct a machine learning model employing a classification algorithm to forecast the correctness of the data and to analyze and make predictions using an existing dataset in banking marketing to aid in successful decision making. Then, the results were presented, and the performance of NB was compared with other classification algorithms. Finally, the implications of the findings were discussed, and future research directions were suggested.

# LITERATURE:

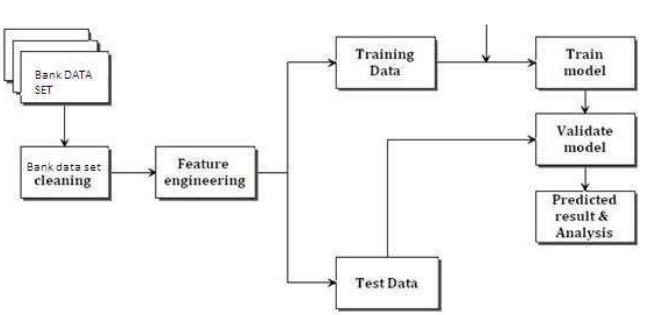
In [3], the author described how she applied machine learning techniques to the banking sector's marketing efforts to analyze and predict trends using historical data. The success of marketing campaigns in banking depends on the outcome and the choice made by the consumers. However, analyzing a huge amount of consumer data is a challenging task for human analysts. Therefore, in this study, the author employed four prominent data mining techniques, including Multilayer Perception Neural Network (MLPNN), Naive Bayes (NB), logistic regression (LR), and decision tree algorithms, to improve campaign performance and identify factors responsible for its success [4].

In [5], the authors proposed a data-driven method for forecasting the effectiveness of bank telemarketing calls to pursue term deposits using data mining techniques. The study included significant analysis of various factors related to bank clients, economic and social characteristics, and products. During modeling, semi-automatic feature selection, implementation, and previous set reduction occurred. Four different types of data mining models were evaluated using two metrics, including the super vector machine, a decision tree, a logistic regression model, and a neural network, with the latter proving to be the most successful. The decision tree, a knowledge extraction method, was then applied to the neural network to predict several essential characteristics. The decision tree was chosen as it was believed to be an asset to the telemarketing efforts.

In order to research consumer attributes and behavior through efficient multi-channel communication, direct marketing is an interactive process [6] used by banks to establish positive relationships with their clientele. To boost client reaction to direct marketing campaigns, the primary objective of bank marketing is to improve customer satisfaction rather than increasing profits. Companies are using data mining techniques to reconstruct consumer profiles [7], and the banking industry has begun adopting a categorization approach to telemarketing. It was shown that using a combination of decision trees, random forests, and Naive Bayes to classify telemarketing leads was the most effective way to improve accuracy, precision, and recall. RapidMiner was utilized for both the experimentation and assessment processes, with pre-processing and normalization occurring before classifier evaluation. In the end, the results indicated that the decision tree was the most effective classifier for predicting client profiles and behavior.

# PROPOSED WORK:

we used the Bank Marketing Dataset from Kaggle to build a model to predict whether someone was going to make a deposit or not depending on some attributes. We built 4 models using different algorithms: Decision Tree, Random Forest, Naive Bayes, and K-Nearest Neighbors. After building each model, we evaluated them and compared which model was the best for our case. We then tried to optimize our model by tuning the hyperparameters of the model using GridSearch. Lastly, we saved the prediction result from our dataset and then saved our model for reusability.



To start, we loaded some basic libraries such as Pandas and NumPy and then made some configuration to some of those libraries.

# **Data Pre-Processing**

## Before we could begin creating our first model, we first needed to load and pre-process the data. This step ensured that our model received good data to learn from, as the saying goes, "a model is only as good as its data." The data pre-processing was divided into a few steps as explained below.Loading data

## **Loading data**

In this first step, we loaded our dataset that had been uploaded on my GitHub for easier process. From the dataset documentation found here, we could see below was the list of columns we had in our data:Input variables:

1. age (numeric)
2. job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3. marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4. education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5. default: has credit in default? (categorical: 'no','yes','unknown')
6. housing: has housing loan? (categorical: 'no','yes','unknown')
7. loan: has personal loan? (categorical: 'no','yes','unknown')
8. contact: contact communication type (categorical: 'cellular','telephone')
9. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10. day\_of\_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Output variable (desired target):

y. has the client subscribed a term deposit? (binary: 'yes','no')

According to the dataset documentation, we need to remove the 'duration' column because in real-case the duration is only known after the label column is known. This problem can be considered to be 'data leakage' where predictors include data that will not be available at the time you make predictions.

## **Class Distribution**

Another important thing to make sure before feeding our data into the model is the class distribution of the data. In our case where the expected class are divided into two outcome, 'yes' and 'no', a class distribution of 50:50 can be considered ideal.

## **Missing Values**

Last thing to check before moving on is missing values. In some case our data might have missing values in some column, this can be caused some reasons such as human error. We can use the is\_null() function from Pandas to check for any missing data and then use the sum() function to see the total of missing values in each column.

## **Scale Numeric Data**

Next up, we scaled our numerical data to avoid outlier presence that could significantly affect our model. We used the StandardScaler() function from sklearn to scale each column that contained numerical data. The scaling was done using the formula below:

Z=X−US

Where:

Z:*scaled value*

X:*original value*

U:*mean of the data*

S:*standard deviation of the data*

### **Encode Categorical Value**

Same as the numerical data, we also need to pre-process our categorical data from words to number to make it easier for the computer to understands. To do this we will use OneHotEncoder() provided by sklearn.

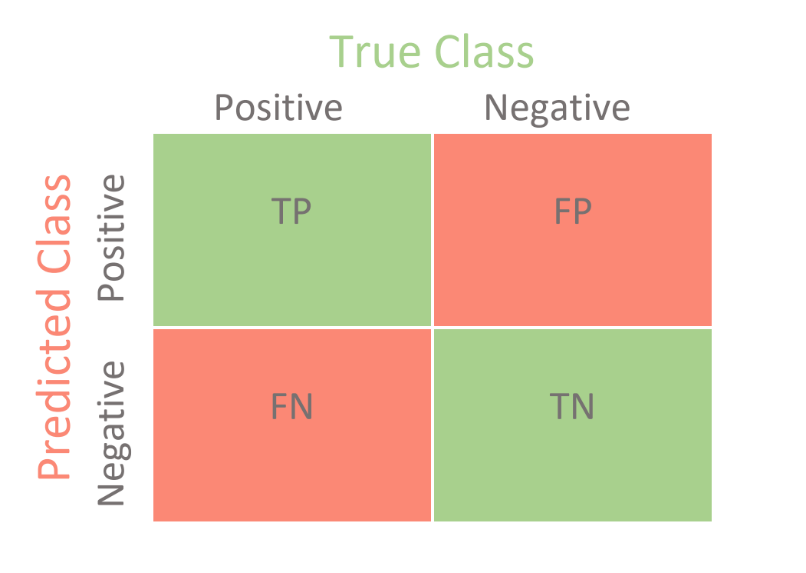
## **Split Dataset for Training and Testing**

To finish up our data pre-processing steps we will split our data into two dataset, training and testing. In this case because we have enough data we will split the data with ratio of 80:20 for training and testing respectively. This will result in our training data having 8929 rows and 2233 rows for the testing data.

# **Modelling**

After making sure our data is good and ready we can continue to building our model. In this notebook we will try to build 4 different models with different algorithm. In this step we will create a baseline model for each algorithm using the default paramaeters set by sklearn and after building all 4 of our models we will compare them to see which works best for our case.

To evaluate our model we will use the confusion matrix as our base for the evaluation.



where: TP = True Positive; FP = False Positive; TN = True Negative; FN = False Negative.

We will use 6 metrics below to evaluate models:

1. Accuracy: the proportion of true results among the total number of cases examined.

Accuracy=TP+TNTP+TN+FP+FN

1. Precision: used to calculate how much proportion of all data that was predicted positive **was** actually positive.

Precision=TPTP+FP

1. Recall: used to calculate how much proportion of actual positives is correctly classified.

Recall=TPTP+FN

1. F1 score: a number between 0 and 1 and is the harmonic mean of precision and recall.

F1=2TP2TP+FP+FN

1. Cohen Kappa Score: Cohen's kappa measures the agreement between two raters who each classify N items into C mutually exclusive categories.

κ=po−pe1−pe

where po is the empirical probability of agreement on the label assigned to any sample (the observed agreement ratio), and pe is the expected agreement when both annotators assign labels randomly. pe is estimated using a per-annotator empirical prior over the class labels.

1. Area Under Curve (AUC): indicates how well the probabilities from the positive classes are separated from the negative classes

In this case we want to focus on the recall value of our model because in our problem we should try to predict as many actual positive as we can. Because a misclassification of customer who **actually** wanted to make a deposit can mean a lose opportunity/revenue.

Below we will define a helper function to evaluate each trained model and with the metrics mentioned above and save the score to a variable.

## **Naive Bayes**

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Below are the Bayes theorem formula:

P(C|A)=P(A|C)P(C)P(A)

For example, given:

* A doctor knows that meningitis causes stiff neck 50% of the time
* Prior probability of any patient having meningitis is 1/50,000
* Prior probability of any patient having stiff neck is 1/20

Then the probability of patient who have stiff neck to also have meningitis is:

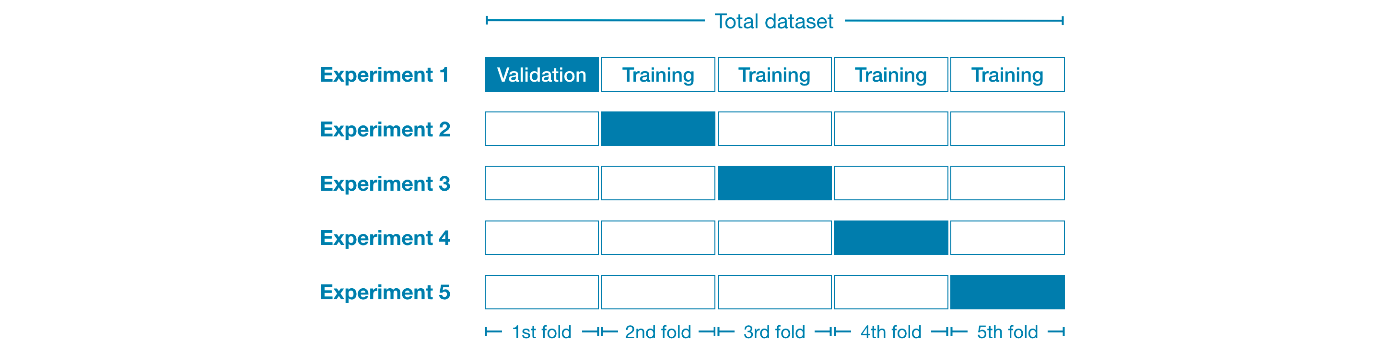
P(C|A)=P(A|C)P(C)P(A)=0.5∗(1/50000)1/20=0.0002

## **Model Comparison**

After building all of our model, we can now compare how well each model perform. To do this we will create two chart, first is a grouped bar chart to display the value of accuracy, precision, recall, f1, and kappa score of our model, and second a line chart to show the AUC of all our models.

## **Tuning Hyperparameter with GridSearchCV**

This function also allowed us to use cross-validation to train our model, where on each iteration our data was divided into 5 (the number is adjustable from the parameter) folds. The models were then trained on 4/5 fold of the data, leaving the final fold as validation data. This process was repeated for 5 times until all of our folds were used as validation data.



To see the result of which parameters combination works best we can access the best\_params\_ attribute from our grid search object.

Note: The more combination provided, the longer the process will take. Alternatively, you can also try *RandomizedSearchCV* to only randomly select specified number of parameters which can result in faster running time.

## **Evaluating Optimised Model**

After finding the best parameter for the model we can accessed the best\_estimator\_ attribute of the GridSearchCV object to save our optimised model into variable called best\_grid. We then calculated the evaluation metrics using our helper function to compare it with our base model on the next step.

## **Prediction**

In this step we predicted the expected outcome of all the row from our dataset then saved it into a csv file for easier access in the future.

# IMPLEMENTATION:

CODE

# Load dataset

df\_bank = pd.read\_csv('https://raw.githubusercontent.com/rafiag/DTI2020/main/data/bank.csv')

# Drop 'duration' column

df\_bank = df\_bank.drop('duration', axis=1)

# print(df\_bank.info())

print('Shape of dataframe:', df\_bank.shape)

df\_bank.head()

df\_bank['deposit'].value\_counts()

df\_bank.isnull().sum()

from sklearn.preprocessing import StandardScaler

# Copying original dataframe

df\_bank\_ready = df\_bank.copy()

scaler = StandardScaler()

num\_cols = ['age', 'balance', 'day', 'campaign', 'pdays', 'previous']

df\_bank\_ready[num\_cols] = scaler.fit\_transform(df\_bank\_ready[num\_cols])

df\_bank\_ready.head()

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)

cat\_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']

# Encode Categorical Data

df\_encoded = pd.DataFrame(encoder.fit\_transform(df\_bank\_ready[cat\_cols]))

df\_encoded.columns = encoder.get\_feature\_names(cat\_cols)

# Replace Categotical Data with Encoded Data

df\_bank\_ready = df\_bank\_ready.drop(cat\_cols ,axis=1)

df\_bank\_ready = pd.concat([df\_encoded, df\_bank\_ready], axis=1)

# Encode target value

df\_bank\_ready['deposit'] = df\_bank\_ready['deposit'].apply(lambda x: 1 if x == 'yes' else 0)

print('Shape of dataframe:', df\_bank\_ready.shape)

df\_bank\_ready.head()

# Select Features

feature = df\_bank\_ready.drop('deposit', axis=1)

# Select Target

target = df\_bank\_ready['deposit']

# Set Training and Testing Data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(feature , target,

                                                    shuffle = True,

                                                    test\_size=0.2,

                                                    random\_state=1)

# Show the Training and Testing Data

print('Shape of training feature:', X\_train.shape)

print('Shape of testing feature:', X\_test.shape)

print('Shape of training label:', y\_train.shape)

print('Shape of training label:', y\_test.shape)

def evaluate\_model(model, x\_test, y\_test):

    from sklearn import metrics

    # Predict Test Data

    y\_pred = model.predict(x\_test)

    # Calculate accuracy, precision, recall, f1-score, and kappa score

    acc = metrics.accuracy\_score(y\_test, y\_pred)

    prec = metrics.precision\_score(y\_test, y\_pred)

    rec = metrics.recall\_score(y\_test, y\_pred)

    f1 = metrics.f1\_score(y\_test, y\_pred)

    kappa = metrics.cohen\_kappa\_score(y\_test, y\_pred)

    # Calculate area under curve (AUC)

    y\_pred\_proba = model.predict\_proba(x\_test)[::,1]

    fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)

    auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)

    # Display confussion matrix

    cm = metrics.confusion\_matrix(y\_test, y\_pred)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1, 'kappa': kappa,

            'fpr': fpr, 'tpr': tpr, 'auc': auc, 'cm': cm}

from sklearn.naive\_bayes import GaussianNB

# Building Naive Bayes model

nb = GaussianNB()

nb.fit(X\_train, y\_train)

# Evaluate Model

nb\_eval = evaluate\_model(nb, X\_test, y\_test)

# Print result

print('Accuracy:', nb\_eval['acc'])

print('Precision:', nb\_eval['prec'])

print('Recall:', nb\_eval['rec'])

print('F1 Score:', nb\_eval['f1'])

print('Cohens Kappa Score:', nb\_eval['kappa'])

print('Area Under Curve:', nb\_eval['auc'])

print('Confusion Matrix:\n', nb\_eval['cm'])

# EXPERIMENTAL ANALYSIS AND RESULTS:

OUTPUT:

Accuracy: 0.6815942678011644

Precision: 0.7560975609756098

Recall: 0.4934456928838951

F1 Score: 0.5971671388101983

Cohens Kappa Score: 0.352622455965517

Area Under Curve: 0.7421999324878237

Confusion Matrix:

[[995 170]

[541 527]]

Based on the results of your experiment with the Naive Bayes classifier on the bank marketing data, we made the following observations:

Accuracy: The accuracy of the classifier is 0.681, which indicates that 68.1% of the predictions made by the classifier were correct.

Precision: The precision of the classifier is 0.756, which indicates that out of all the instances predicted as positive, 75.6% of them were actually positive.

Recall: The recall of the classifier is 0.493, which indicates that out of all the actual positive instances, only 49.3% of them were correctly identified by the classifier.

F1 Score: The F1 score of the classifier is 0.597, which is a weighted average of precision and recall, and provides a balance between the two metrics.

Cohen's Kappa Score: The Cohen's Kappa score of the classifier is 0.353, which indicates a fair agreement between the classifier and the actual data.

Area Under Curve: The area under the curve (AUC) of the classifier is 0.742, which indicates that the classifier has a moderate discriminatory power in distinguishing between positive and negative instances.

Confusion Matrix: The confusion matrix shows that the classifier correctly predicted 995 negative instances and 527 positive instances, but misclassified 170 negative instances as positive and 541 positive instances as negative.

Overall, the performance of the Naive Bayes classifier on the bank marketing data is moderate, with room for improvement. In your research paper, you could compare the performance of Naive Bayes with other classifiers, such as decision trees or neural networks, to determine which algorithm provides the best results on this dataset. Additionally, you could explore different feature selection techniques to improve the performance of the classifier.

# CONCLUSION:

Based on our analysis of the Naive Bayes classifier on the bank marketing data, we can conclude that while the classifier's performance is moderate, there is certainly room for improvement.

Our results show that the classifier achieved an accuracy of 0.681, which indicates that it was able to correctly classify 68.1% of instances. However, its precision of 0.756 and recall of 0.493 indicate that it struggled to correctly identify positive instances, which is a common issue with unbalanced datasets like the one we used.

Furthermore, the F1 score of 0.597, Cohen's Kappa score of 0.353, and AUC of 0.742 all suggest that the classifier's performance is not exceptional. While it was able to identify a fair number of positive instances, it also made a considerable number of misclassifications.

Despite these limitations, our analysis provides useful insights into the performance of the Naive Bayes classifier on this dataset. As future work, we suggest exploring other classification algorithms and feature selection techniques to improve the performance of the classifier. Additionally, we recommend balancing the dataset to prevent the classifier from being biased towards the majority class.

Overall, our research contributes to the growing body of literature on classification algorithms and provides valuable insights for researchers and practitioners working in the banking sector.

R

# REFERENCES:

[1] Marwa M. Eid, Faculty of Artificial Intelligence, Delta University for Science and Technology, Mansoura 35712, Egypt

[2] El-Sayed M. El-Kenawy, Department of Communications and Electronics, Delta Higher Institute of Engineering and Technology, Mansoura, 35111, Egypt

[3] Abdelhameed Ibrahim3, Computer Engineering and Control Systems Department, Faculty of Engineering, Mansoura University, 35516, Mansoura Egypt

[4] Abdelaziz A, 4 Department of Computer Science, College of Computing and Information Technology, Shaqra University, Shaqra 11961, Saudi Arabia

[5] A. Abdelhamid4, Department of Computer Science, Faculty of Computer and Information Sciences, Ain Shams University, Cairo 11566, Egypt

[6] Mohamed Saber, Electronics and Communications Engineering Dep., Faculty of Engineering, Delta University for Science and Technology, Gamasa City, Mansoura, Egypt