Smeet Shah

Use Naive Bayesian classification in the finance domain to predict loan default based on financial and personal attributes

📚 Description of Problem Statement:

The problem involves using **Naive Bayesian classification** to predict whether a loan applicant will default, based on a set of financial and personal attributes, such as income, credit history, employment status, and debt-to-income ratio. The goal is to classify potential borrowers into two categories: those who are likely to default and those who are not. The Naive Bayes algorithm is chosen because of its simplicity and efficiency in handling large datasets, assuming that the features are conditionally independent of each other. This model can help financial institutions make informed lending decisions.

📚 Theory :

Definition of Naive Bayes Classification:

Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It assumes that the features (or attributes) are independent of each other given the class label, hence the term "naive." Despite this simplifying assumption, Naive Bayes classifiers often perform surprisingly well, especially in text classification and domains where the independence assumption is not severely violated.

Bayes' Theorem is expressed as:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

- P(C|X) is the posterior probability of class C given the input features X,
- P(X|C) is the likelihood of features X given the class C,
- P(C) is the prior probability of the class C,
- P(X) is the evidence or marginal probability of X.

In a classification setting, we calculate the posterior probabilities for each possible class and assign the class with the highest probability to the instance.

2. Why is Naive Bayes Used?

Naive Bayes is used in many domains, including finance, due to its simplicity, efficiency, and ability to handle large datasets. In finance, predicting loan defaults is a critical task where lenders want to assess the risk associated with lending to a potential borrower. Some reasons for using Naive Bayes in finance include:

- **Speed and Scalability**: It works well with large datasets, making it suitable for high-volume financial data.
- **Probabilistic Interpretation**: It provides probability estimates for each class, which can help in decision-making for financial institutions.

- **Robustness with Noise**: Naive Bayes performs well even when the independence assumption is violated to some degree.

3. Loan Default Prediction using Naive Bayes:

In finance, predicting loan default is a common application of machine learning. Loan default prediction involves using financial and personal attributes to assess whether a borrower is likely to default on a loan. This can help financial institutions in deciding whether to approve a loan and what interest rate to offer.

4. Steps to Apply Naive Bayes for Loan Default Prediction:

- **Step 1: Data Collection** Collect historical loan data with attributes such as income, credit score, loan amount, and default status (target variable).
- **Step 2: Data Preprocessing** Clean the data by handling missing values, encoding categorical features (e.g., employment status, homeownership status), and normalizing numerical features if necessary.
- **Step 3: Calculate Prior Probabilities** Compute the prior probability for each class (default/no default) from the historical data.

$$P(\text{Default}) = \frac{\text{Number of defaults}}{\text{Total number of loans}}$$

$$P(\text{No Default}) = \frac{\text{Number of non-defaults}}{\text{Total number of loans}}$$

- **Step 4: Calculate Likelihood** For each feature, calculate the likelihood for each class. For example, if "income" is one of the attributes:

$$P(\text{Income}|\text{Default}) = \frac{\text{Number of people with income in a specific range who defaulted}}{\text{Total number of defaulters}}$$

Do this for all features.

- **Step 5: Apply Bayes' Theorem** Use Bayes' Theorem to calculate the posterior probability of the borrower defaulting based on the given attributes. The class with the highest posterior probability is assigned to the borrower.
- **Step 6: Model Evaluation** Use metrics such as accuracy, precision, recall, to evaluate the model's performance on a test dataset.

Data Sources: https://www.kaggle.com/datasets/nikhil1e9/loan-default

Detail about data taken it's attributes with description

The dataset contains information about loan applicants and their status regarding loan defaults.

1. **LoanID**: A unique identifier for each loan application.
2. **Age**: The age of the applicant.
3. **Income**: The annual income of the applicant (in local currency).
4. **LoanAmount**: The amount of the loan requested by the applicant.
5. **CreditScore**: The applicant's credit score, indicating creditworthiness.
6. **MonthsEmployed**: The number of months the applicant has been employed.
7. **NumCreditLines**: The number of credit lines (accounts) the applicant has open.
8. **InterestRate**: The interest rate applied to the loan.
9. **LoanTerm**: The duration of the loan in months.
10. **DTIRatio**: Debt-to-Income (DTI) ratio, representing the applicant's debt obligations compared to their income.
11. **Education**: The educational qualification of the applicant (e.g., Bachelor's, Master's, High School).
12. **EmploymentType**: The type of employment the applicant holds (e.g., Full-time, Unemployed).
13. **MaritalStatus**: The marital status of the applicant (e.g., Married, Divorced).
14. **HasMortgage**: Whether the applicant has an existing mortgage (Yes/No).
15. **HasDependents**: Whether the applicant has dependents (Yes/No).
16. **LoanPurpose**: The purpose of the loan (e.g., Auto, Business, Other).
17. **HasCoSigner**: Whether the loan has a co-signer (Yes/No).
18. **Default**: The target variable indicating whether the applicant defaulted on the loan (1 = Default, 0 = No Default).

Code & Output:

```
import pandas as pd

# Load the dataset loan_data = pd.read_csv('/kaggle/input/loan-
default/Loan_default.csv')

# Display basic info and first few rows
loan_data.info()
loan_data.head()
```

```
RangeIndex: 255347 entries, 0 to 255346
Data columns (total 18 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   LoanID          255347 non-null  object
 1   Age             255347 non-null  int64
 2   Income          255347 non-null  int64
 3   LoanAmount      255347 non-null  int64
 4   CreditScore     255347 non-null  int64
 5   MonthsEmployed  255347 non-null  int64
 6   NumCreditLines  255347 non-null  int64
 7   InterestRate    255347 non-null  float64
 8   LoanTerm        255347 non-null  int64
 9   DTIRatio        255347 non-null  float64
 10  Education       255347 non-null  object
 11  EmploymentType  255347 non-null  object
 12  MaritalStatus   255347 non-null  object
 13  HasMortgage     255347 non-null  object
 14  HasDependents   255347 non-null  object
 15  LoanPurpose     255347 non-null  object
 16  HasCoSigner     255347 non-null  object
 17  Default         255347 non-null  int64
dtypes: float64(2), int64(8), object(8)
memory usage: 35.1+ MB
```

	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	Education
0	I38PQUQS96	56	85994	50587	520	80	4	15.23	36	0.44	Bachelor's
1	HPSK72WA7R	69	50432	124440	458	15	1	4.81	60	0.68	Master's
2	C1OZ6DPJ8Y	46	84208	129188	451	26	3	21.17	24	0.31	Master's
3	V2KKSFM3UN	32	31713	44799	743	0	3	7.07	24	0.23	High School
4	EY08JDHTZP	60	20437	9139	633	8	4	6.51	48	0.73	Bachelor's

EmploymentType	MaritalStatus	HasMortgage	HasDependents	LoanPurpose	HasCoSigner	Default
Full-time	Divorced	Yes	Yes	Other	Yes	0
Full-time	Married	No	No	Other	Yes	0
Unemployed	Divorced	Yes	Yes	Auto	No	1
Full-time	Married	No	No	Business	No	0
Unemployed	Divorced	No	Yes	Auto	No	0

Drop the LoanID column as it's not useful for prediction

loan_data_cleaned = loan_data.drop('LoanID', axis=1)

Display the updated data to confirm loan_data_cleaned.head()

	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate
0	56	85994	50587	520	80	4	15.23
1	69	50432	124440	458	15	1	4.81
2	46	84208	129188	451	26	3	21.17
3	32	31713	44799	743	0	3	7.07
4	60	20437	9139	633	8	4	6.51

---

from sklearn.preprocessing import LabelEncoder

Initialize the label encoder label_encoders

= {}

Loop through all columns with object data type and apply Label Encoding

for column in loan_data_cleaned.select_dtypes(include=['object']).columns:

le = LabelEncoder()

    loan_data_cleaned[column] = le.fit_transform(loan_data_cleaned[column])

label_encoders[column] = le # Store the encoder for later use (if needed)

Display the first few rows to verify the encoding

loan_data_cleaned.head()

oanTerm	DTIRatio	Education	EmploymentType	MaritalStatus	HasMortgage	HasDependents	LoanPurpose	HasCoSigner	Default
36	0.44	0	0	0	1	1	4	1	0
60	0.68	2	0	1	0	0	4	1	0
24	0.31	2	3	0	1	1	0	0	1
24	0.23	1	0	1	0	0	1	0	0
48	0.73	0	3	0	0	1	0	0	0

```
# Split the data into features (X) and target (y)
X = loan_data_cleaned.drop('Default', axis=1)  # Features y
= loan_data_cleaned['Default']  # Target

# Display the shape of the features and target to confirm
print(X.shape, y.shape)
```

```
(255347, 16) (255347,)
```

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Display the shape of the training and testing sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(178742, 16) (76605, 16) (178742,) (76605,)
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Use the same scaler to transform the test data
X_test_scaled = scaler.transform(X_test)

# X_train_scaled and X_test_scaled are ready for model training

from sklearn.naive_bayes import GaussianNB from sklearn.metrics import
accuracy_score, classification_report, confusion_matrix

# Initialize the Gaussian Naive Bayes model
nb_model = GaussianNB()

# Train the model on the scaled training data
nb_model.fit(X_train_scaled, y_train)

# Make predictions on the test data y_pred
= nb_model.predict(X_test_scaled)

# Evaluate the model accuracy =
accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred) class_report =
classification_report(y_test, y_pred) # Display the
evaluation metrics print(f"Accuracy: {accuracy *
100:.2f}%") print("Confusion Matrix:")
```

print(conf_matrix) print("Classification Report:")

print(class_report)

```
Accuracy: 88.45%
Confusion Matrix:
[[67609    72]
 [ 8778   146]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     67681
           1       0.67      0.02      0.03      8924

    accuracy                           0.88     76605
   macro avg       0.78      0.51      0.49     76605
weighted avg       0.86      0.88      0.83     76605
```

---

import matplotlib.pyplot as plt import

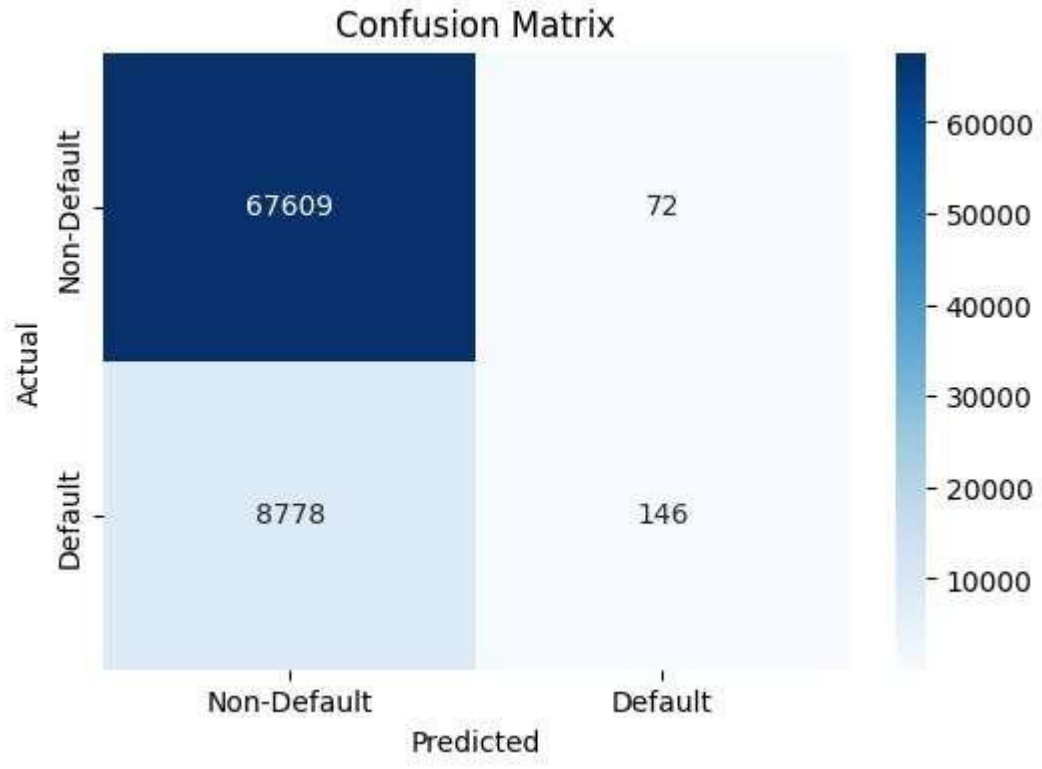
seaborn as sns

from sklearn.metrics import roc_curve, auc

Confusion Matrix Heatmap

plt.figure(figsize=(6, 4))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Default', 'Default'], yticklabels=['Non-Default', 'Default']) plt.xlabel('Predicted') plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()



- **True Negatives (TN)**: **67,609**

- These are the instances where the actual value is **0** (No Default) and the model predicted **0** (No Default).
- The model correctly predicted **67,609 loans** that did not default. This is a good match.

- **False Positives (FP)**: **72**

- These are the instances where the actual value is **0** (No Default), but the model predicted **1** (Default).
- The model incorrectly predicted **72 loans** would default when they actually did not. This is a mismatch.

- **False Negatives (FN)**: **8,778**

- These are the instances where the actual value is **1** (Default), but the model predicted **0** (No Default).
- The model missed **8,778 actual defaults**. This is a significant mismatch and indicates poor performance in capturing defaults.

- **True Positives (TP)**: **146**

- These are the instances where the actual value is **1** (Default) and the model predicted **1** (Default).
- The model correctly predicted **146 loans** that did default. This is a match, but the number is relatively low compared to the total defaults.

Conclusion:

Naive Bayes classification is a powerful tool for predicting loan defaults in the finance domain. Its efficiency, simplicity, and ability to handle large datasets make it particularly useful for realtime applications like loan approvals. Despite its limitations, Naive Bayes can provide valuable insights into risk assessment, helping financial institutions make informed lending decisions while managing risk effectively. As financial data continues to grow, utilizing models like Naive Bayes can help institutions stay competitive and responsive to market conditions. By leveraging Naive Bayes in loan default prediction, financial institutions can enhance their risk assessment processes, reduce losses from defaults, and ultimately make more strategic lending decisions.