Credit Score Classification Using Machine Learning

By: Vatsal Sivaratri

Problem Statement and Motivation

Goal: Develop a predictive, robust model(s) for classifying individuals' credit scores.

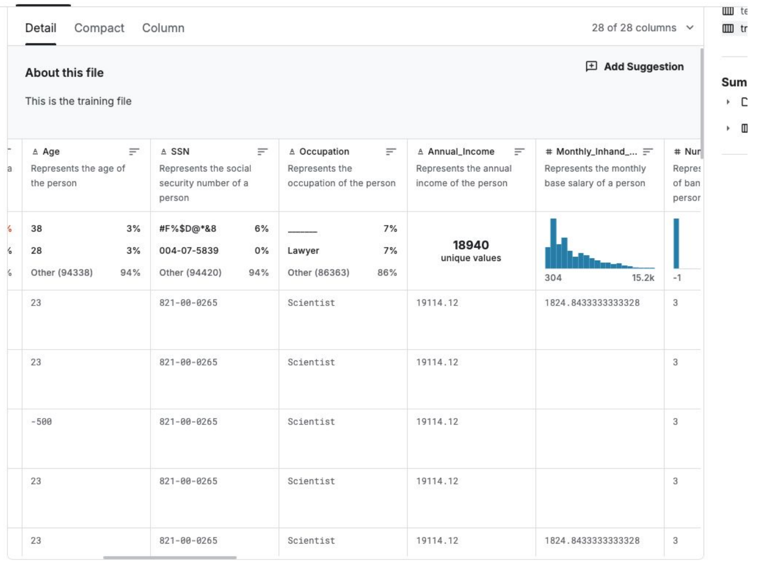
Motivation: There is a growing need for transparency and accuracy in evaluating trustworthiness of Credit Card users. A strong relationship between banks and their customers is necessary for a productive economy. The hope is to use past data on credit card holders to predict the likelihood of a good credit score in new customers, by considering a variety of attributes.

Dataset Source & Description

- **Source:** Kaggle Card ScoreDataset
- **Key Details:** Training dataset consists of 100,000 instances and 27 features, including both numerical and categorical.
- **Features:** Age, Occupation, Annual_Income, Monthly_Inhand_Salary, Outstanding_Debt, Credit_Mix, etc.
- Data was already split into training and testing set, but additional preprocessing was required to clean and handle inconsistencies. Only training set was used as it had enough samples.
- 3 different classes of Credit_Score: Good, Poor, and Standard

Data Types and Important Features

- **Numerical Data:** Features like Annual_Income, Outstanding_Debt, and Credit_Utilization_Ratio
- **Categorical Data:** Features like Occupation, Credit_Mix, and Payment_Behaviour.
- Many of the features were annoyingly represented as objects, meaning they had to be converted later in Python for further analysis
- Understanding types of features were crucial in determining the preprocessing techniques.



Initial Issues Found in Data

- **Outliers:** values such as age had anomalies like negative or extremely large values (i.e. -500 and >8900 for age)
    - These are a part of any legitimate dataset, and represent a portion of people who chose not to answer truthfully or were incorrectly recorded
- **Missing Data:** Several columns such as Monthly_Inhand_Salary had missing values that could skew the predictions if left unaddressed
- Trailing underscores were found in several columns

```
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           98800 non-null   object
 27  Credit_Score              100000 non-null  object
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

```
data['Annual_Income'].unique()
```

[308] ✓ 0.0s Python

```
array(['19114.12', '34847.84', '34847.84_', ..., '20002.88', '39628.99'
       '39628.99_'], dtype=object)
```

Outlier Detection and Handling

- Each feature was analyzed manually for extreme or blatantly incorrect values with simple python functions like min() and max(). They were filtered out the Pandas Dataframe with simple logical statements
- For quantitative continuous variables, additional outliers were detected and removed using Interquartile Range (IQR)

```
…
    age = pd.to_numeric(data['Age'], errors='coerce')

    wrong_ages = age[(age == -500) | (age>995) |
(age<18)].index

    data = data.drop(wrong_ages)

    data['Age'] = pd.to_numeric(data['Age'],
errors='coerce')
…

Q1 = data[feature].quantile(0.25) Q3 =
data[feature].quantile(0.75) IQR = Q3 - Q1 data =
data[~((data[feature] < (Q1 - 1.5 * IQR)) |
(data[feature] > (Q3 + 1.5 * IQR)))]
```

Missing Data Handling

For numerical columns, missing data was filled using the **mean.** For categorical columns, missing values were filled using the mode.

```
mean_columns = ['Monthly_Inhand_Salary', 'Amount_invested_monthly', 'Monthly_Balance', 'Changed_Credit_Limit']
mode_columns = ['Num_of_Delayed_Payment', 'Num_Credit_Inquiries']
categorical_mode = ['Payment_Behaviour']

for column in mean_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')

    data[column].fillna(data[column].mean(), inplace=True)
for column in mode_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')

    data[column].fillna(data[column].mode()[0], inplace=True)
```

Encoding Categorical Variables

Label Encoding: Categorical variables were converted into numerical format for models to use using Label Encoding.

```
toEncode = ['Month', 'Occupation', 'Credit_Mix', 'Payment_Behaviour', 'Credit_Score']
label_encoder = LabelEncoder()

for column in toEncode:
    data[column] = label_encoder.fit_transform(data[column])
```

***The classes were also converted like this. Now, ‘Good’ is 0, ‘Poor’ is 1, and ‘Standard’ is 2.**

Feature Engineering

Loan Columns: The original dataset had a column for Type_of_Loan that combined multiple loan types (Auto Loan, Credit-Builder Loan, Debt Consolidation Loan, Home Equity Loan, Mortgage Loan, No Loan, Not Specified, Payday Loan, Personal Loan, Student Loan). This was split into multiple binary columns

```
unique_loan_types = ['Auto Loan', 'Mortgage Loan', 'Personal Loan', ...]
for loan_type in unique_loan_types:
    cleaned_loan_type = loan_type.replace(' ', '_').lower()
    data[cleaned_loan_type] = data['Type_of_Loan'].apply(lambda x: loan_type in x if
isinstance(x, str) else False)
```

Then ‘Auto Loan, Mortgage Loan’ **=>** {...,Auto Loan: 1, Mortgage Loan: 1….}

Feature Selection Techniques

- **CorrelationAttributeEval:** Method evaluates correlation between individual attributes and the class label (Pearson’s Correlation Coefficient). Attributes with high correlation to the class are selected (cutoff=0.10)
- **ReliefAttributeEval:** Algorithm estimate feature importance based on how well feature can distinguish between instances that are near each other (cutoff=0.02)
- **ClassifierAttributeEval:** Ranks features based on how well they contribute to classifier performance (I used RandomForest). Features like Annual_Income and Outstanding_Debt got prioritized here.
- **CfsSubsetEval:** Selects group of features that work well together. Identified attributes like Num_Bank_Accounts adn Credit_Mix being most relevant for predicting credit scores.
- **Individually Selected Features:** 'Annual_Income','Monthly_Inhand_Salary','Num_of_Loan', 'Outstanding_Debt','Credit_Utilization_Ratio','Credit_History_Age', 'Num_Credit_Card','Num_of_Delayed_Payment','Interest_Rate','Total_EMI_per_month'

CorrelationAttributeEval

Selected Features:

0.18151 12 Changed_Credit_Limit

0.17729 11 Num_of_Delayed_Payment

0.17391 18 Payment_of_Min_Amount

0.16813 6 Num_Bank_Accounts

0.13492 14 Credit_Mix

0.11204 8 Interest_Rate

0.10648 7 Num_Credit_Card

ReliefAttributeEval

Selected Features:

0.024477   2 Age

-0.021545   18 Payment_of_Min_Amount

-0.023212   14 Credit_Mix

-0.032999   21 Payment_Behaviour

-0.060802   1 Month

ClassifierAttributeEval

Selected Features

0.1879941	4	Annual_Income
0.1812442	15	Outstanding_Debt
0.1583674	19	Total_EMI_per_month
0.1448428	5	Monthly_Inhand_Salary
-0.1479401	20	Amount_invested_monthly
-0.1575615	22	Monthly_Balance
-0.1602462	16	Credit_Utilization_Ratio

CfsSubsetEval

Selected Features:

Num_Bank_Accounts

Num_of_Delayed_Payment

Changed_Credit_Limit

Credit_Mix

Payment_of_Min_Amount

Models

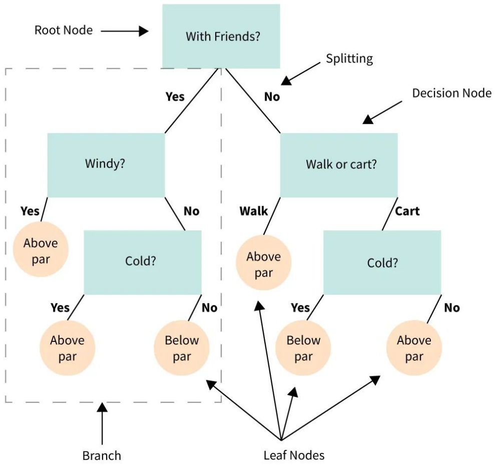
Models Evaluated: Decision Tree, Random Forest, Logistic Regression, Naive Bayes,Support Vector Machines (SVM). Evaluated using Accuracy, True Positive Rates (TPR), False Positive Rates (FPR), and ROC-AUCs

```
models = {
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Naive Bayes": GaussianNB(),
    "SVM": SVC(probability=True)
}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

Decision Tree Classifier

Decision Trees splits data into branches based on feature values to classify data points with a sequence of actions. Structured with a root node, decision nodes, and leaf nodes.

- Tree begins with root node and splits data at each node based on feature that minimizes impurity
- At each node, the algorithm selects the split that yields the purest possible branches



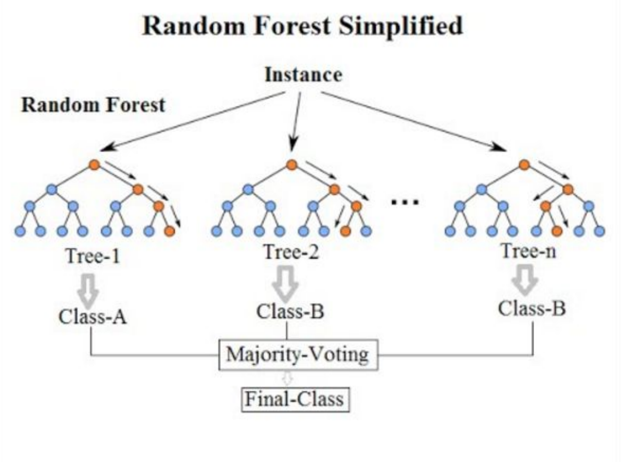
Strengths: Interpretable and easy to visualize, can handle both numerical and categorical data.
Weaknesses: Prone to overfitting, and sensitive to small changes in data.

Random Forest Classifier

Random Forests combine multiple decision trees on random data subsets and combine their results for a more generalized output

- Each tree is trained on a random sample of the data
- Random feature selection occurs for each split point
- Predictions from all trees are averaged

Strengths: Reduces overfitting by averaging multiple models, and works well with high-dimensional data
Weaknesses: Computationally intensive, low interpretability

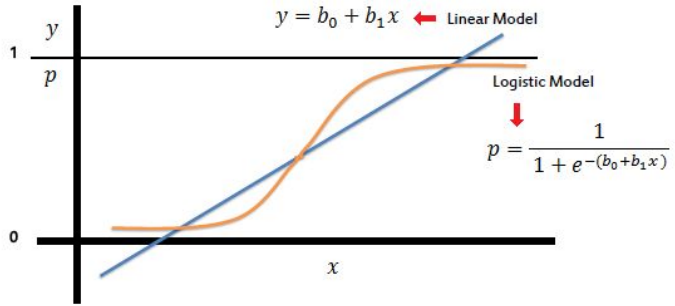


Logistic Regression

Logistic Regression is a linear model for classification, usually effective in binary cases. It estimate the probability of each class with a sigmoid function.

- Fits a linear equation to predict the probability of a data point belonging to a class
- Uses sigmoid to transform predictions into probabilities

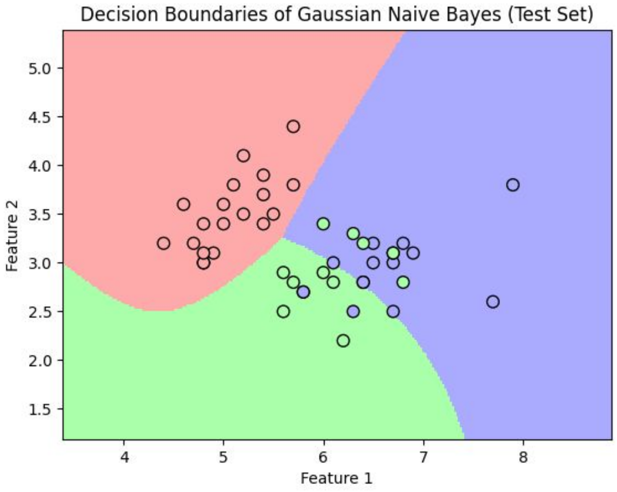
Strengths: Simple and effective for linearly separable data.
Weaknesses: Can't model nonlinear relationships and sensitive to collinearity and outliers.



Naive Bayes

Naive Bayes is based on **Bayes' theorem,** and assumes all features are independent given the class label.

- Calculates probability of each class based on each feature independently
- Assumes independence between features
- Predictions are based on the class with the highest probability.

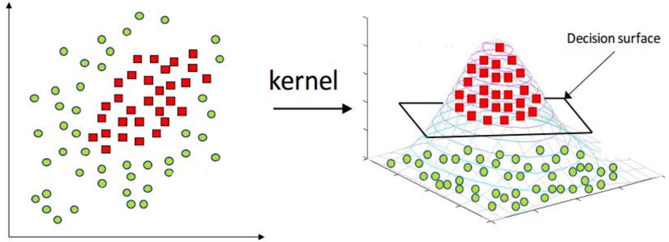


Strengths: Fast and efficient
Weaknesses: Independence assumption can limit accuracy by a LOT.

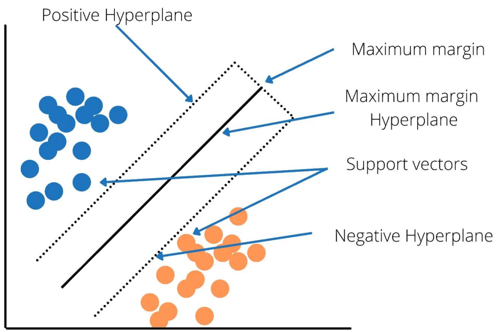
Support Vector Machine (SVM)

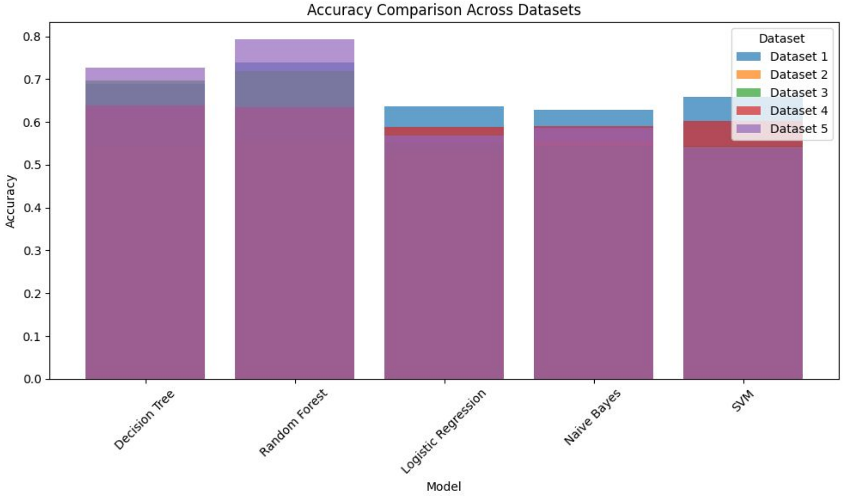
Support Vector Machine (SVM) finds a hyperplane that best separates classes by maximizing margin between classes' boundary points.

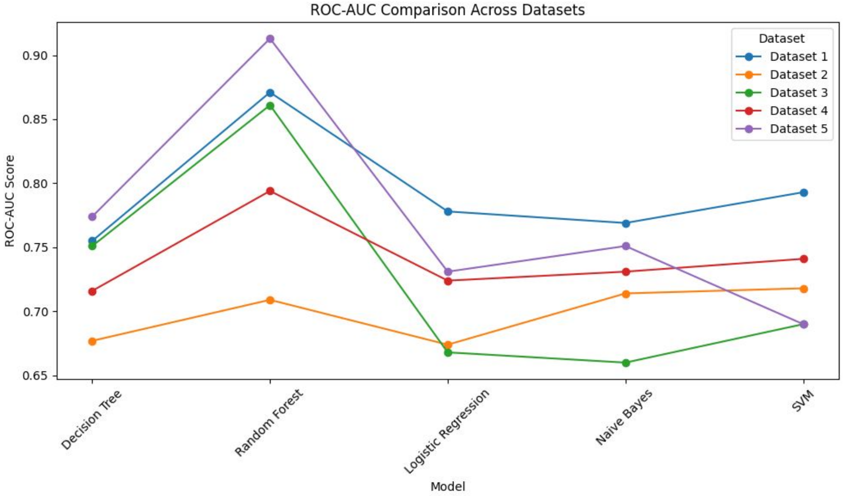
- For linearly separable data SVM finds a linear boundary
- For non-linear data, SVM uses kernels to project data into higher dimensions.

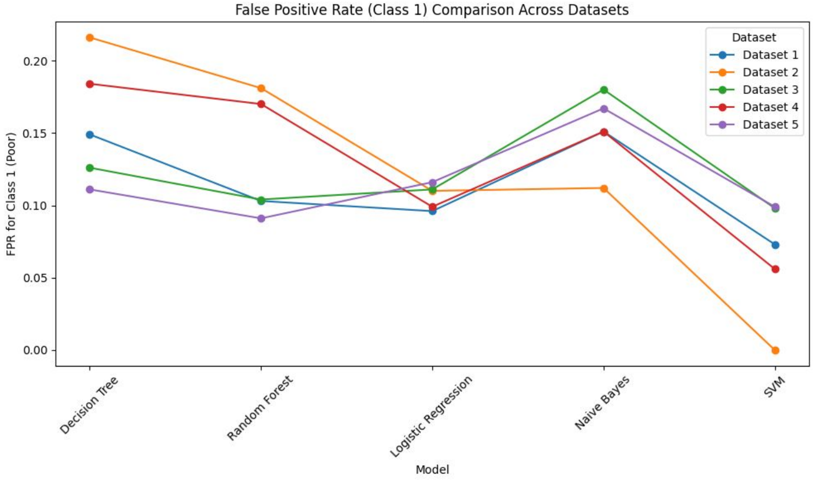


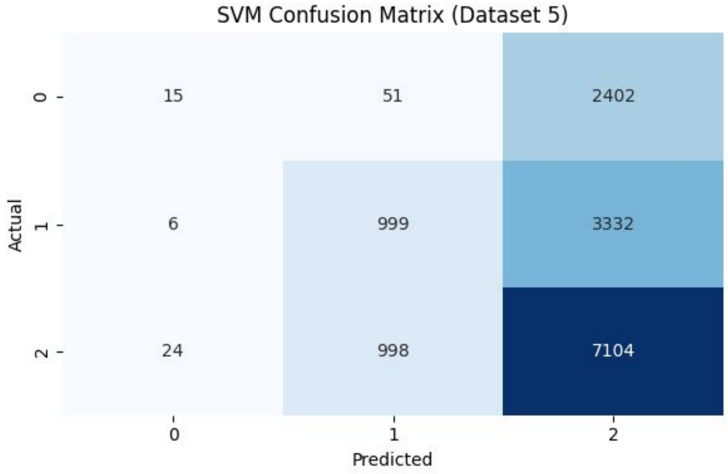
Strengths: Effective in high dimensional spaces. Flexible with various kernel functions; allows for both linear and non-linear classification
Weaknesses: Computationally intensive with large datasets, less interpretable




Accuracy Comparison Across Datasets
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Dataset
Dataset 1
Dataset 2
Dataset 3
Dataset 4
Dataset 5
Decision Tree
Random Forest
Logistic Regression
Naive Bayes
SVM
Model


ROC-AUC Comparison Across Datasets
ROC-AUC Score
0.65
0.70
0.75
0.80
0.85
0.90
Dataset
Dataset 1
Dataset 2
Dataset 3
Dataset 4
Dataset 5
Decision Tree
Random Forest
Logistic Regression
Naive Bayes
SVM
Model


False Positive Rate (Class 1) Comparison Across Datasets
FPR for Class 1 (Poor)
0.00
0.05
0.10
0.15
0.20
Dataset
Dataset 1
Dataset 2
Dataset 3
Dataset 4
Dataset 5
Decision Tree
Random Forest
Logistic Regression
Naive Bayes
SVM
Model


SVM Confusion Matrix (Dataset 5)
Actual
0
1
2
15
51
2402
6
999
3332
24
998
7104
0
1
2
Predicted

Discussion

- **Random Forest** performed best, likely due to the ensemble approach which reduced the variance and captured more complex relationships
- **Decision Tree** was decent, but it is prone to overfitting in tasks like these due to the high dimensionality of the data
- **Logistic Regression** and **Naive Bayes** performed poorly due to underlying assumptions of linearity and feature independence, which clearly doesn’t hold for this dataset.
- Future work can focus on hyperparameter tuning, further analyzing those non-linear features, and most importantly, introducing neural network models to significantly improve model performance and address class imbalance.

Thanks!