Q1 Machine Learning Project Report: Credit Score Classification

By: Vatsal Sivaratri

9/18/2024

Table of Contents

1 Statement/Project Goal

The goal of this project is to develop a predictive, robust model(s) for classifying individuals' credit scores. The motivation behind this project is the 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.

2 Description of Dataset

The data was found on Kaggle from a competition hosted by the website themselves. This dataset has conveniently been split into training and testing datasets. The training dataset contains 100,000 instances and 27 attributes (Dimension – 28). The attributes include both numerical data (e.g., monthly in-hand salary, outstanding debt) and categorical data (e.g., occupation, credit mix). Here's a list of the key attributes:

Age: Age of individual
Occupation: Profession of Individual
Annual_Income: Annual earnings of individual
Monthly_Inhand_Salary: Amount individual receives per month.
Num_Bank_Accounts: Number of bank accounts the individual holds
Outstanding_Debt: Amount of unpaid debt
Credit_Utilization_Ratio: The ratio of the credit limit utilized
Credit_History_Age: The duration of the individual's credit history
Credit_Mix: Information on the mix of credit types (e.g., Good, Bad).
Monthly_Balance: The remaining balance at the end of each month after all payments.

There are some interesting observations, including potential outliers in the age column (like negative values for age), repeated values in columns that are supposed to be unique (SSN), and missing values in some columns.

3 Methods and Materials

Data Preprocessing:

When we first start with our dataset, we have a total of 27 features, 1 class variable, with 100,000 instances.

```
 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
```

1. **Removing any Overtly Useless Columns:** Ex. A person's SSN is useless in predicting their credit score without any other relevant information given by it (and even then it shouldn't be needed as a feature itself)=> Go through all features and see which ones aren't productive and need to be removed.

```
useless_columns = ['ID', 'Customer_ID', 'Name', 'SSN']
data.drop(useless_columns, axis=1, inplace=True)
```

2. **Handling Underscore Issue:** In this dataset, there appears to be a continued problem of instances have an extra underscore, which prevents further analysis.

```
data['Annual_Income'].unique()
[308]  ✓ 0.0s                                                   Python
...  array(['19114.12', '34847.84', '34847.84_', ..., '20002.88', '39628.99'
       '39628.99_'], dtype=object)
```

```
def remove_trailing_underscore(df, column_name):
    def clean_value(value):
        if isinstance(value, str) and value.endswith('_'):
            return value[:-1]
        return value

    df[column_name] = df[column_name].apply(clean_value)

    return df

for column in data.columns:
    data = remove_trailing_underscore(data, column)
```

3. **Outlier Detection and Removal:** Certain variables like "Age" and "Monthly_Balance" can have extreme values that are outliers and bad representations of other cars with similar specifications. These will need to be identified and removed, both manually and with IQR.

Graph distributions of features to see any apparent outliers:

```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_feature_distributions(data, plot_type='hist'):
    num_columns = len(data.columns)

    plt.figure(figsize=(16, num_columns * 4))

    for i, column in enumerate(data.columns, 1):
        plt.subplot(num_columns, 1, i)

        if plot_type == 'hist':
            sns.histplot(data[column], kde=True, bins=100)
            plt.title(f'Distribution of {column}')
        elif plot_type == 'box':
            sns.boxplot(x=data[column])
            plt.title(f'Boxplot of {column}')

        plt.tight_layout()

    plt.show()

plot_feature_distributions(data, plot_type='hist')
```

Output:
https://drive.google.com/file/d/1r5UP0r1FlC_KLzjFjZdry6x4WfDCn0qW/view?usp=sharing

Remove Filler Outliers: Datasets like these often have people inputting obviously wrong data as placeholders. These methods take care of those issues.

```
def process_age(data):
   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')

   return data
data = process_age(data)

def process_ir(data):
   ir = pd.to_numeric(data['Interest_Rate'], errors='coerce')

   wrong_irs = ir[(ir <0) | (ir>100)].index

   data = data.drop(wrong_irs)

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

   return data
data = process_ir(data)

def convert_credit_history_age_to_months(data):
   def convert_to_months(value):
       if pd.isna(value):
           return None
       parts = value.split('and')
       years = int(parts[0].strip().split()[0])
       months = int(parts[1].strip().split()[0])
```

```
        return years * 12 + months


    data['Credit_History_Age'] =
data['Credit_History_Age'].apply(convert_to_months)

    data['Credit_History_Age'].fillna(data['Credit_History_Age'].mean(),
inplace=True)

    return data
data = convert_credit_history_age_to_months(data)

def process_delayed_payment(data):
    delayed_payment = pd.to_numeric(data['Num_of_Delayed_Payment'],
errors='coerce')

    wrong_irs = delayed_payment[((delayed_payment < 0))].index

    data = data.drop(wrong_irs)

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

    return data
data = process_delayed_payment(data)


def process_amount_invested_monthly(data):

    data['Amount_invested_monthly'] =
data['Amount_invested_monthly'].replace('__10000_', pd.NA)

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

    return data

data = process_amount_invested_monthly(data)


def process_payment_behavior(data):
```

```
    data['Payment_Behaviour'] = data['Payment_Behaviour'].replace('!@9#%8', '0')

    return data
data = process_payment_behavior(data)
```

For quantitative continuous features, also remove outliers with IQR:

```
features_to_process = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary',
                      'Outstanding_Debt', 'Credit_Utilization_Ratio',
                      'Total_EMI_per_month', 'Amount_invested_monthly']

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


    data[feature].fillna(data[feature].mean(), inplace=True)

    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)))]
```

4. **Fix Loan Feature Representations:** Loans are a very important feature in determining what kind of credit score a person has. However, the way loans are currently represented in the dataset combines all of them as one feature, making it hard to represent meaningfully to the model (Too many combinations of loan types). Thus, split this feature into 10 other features, 1 for each possible category:

```
def add_loan_type_columns(data):
    unique_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']

    for loan_type in unique_loan_types:

        cleaned_loan_type = loan_type.replace(' ', '_').replace('-',
'_').lower()

        data[cleaned_loan_type] = data['Type_of_Loan'].apply(lambda x:
x.count(loan_type) if isinstance(x, str) else 0)
    data.drop(['Type_of_Loan'], axis=1, inplace=True)
```

```
    return data
data = add_loan_type_columns(data)
```

5. **Handle Missing Data:** As mentioned before, several columns contain missing values. How to fill them depends on the feature being dealt with. For example, the column "Monthly_Inhand_Salary" has missing values, and since this is a numerical, continuous feature, we may best fill them using the mean of the column. For "Num_Credit_Inquiries", "Occupation" and other discrete quantitative or qualitative features, values will be filled with 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)
```

6. **Encoding Categorical Variables:** Columns like "Occupation" and "Credit_Mix" need to be encoded as numerical values for the model. This might involve one-hot encoding or label encoding.

```
toEncode = ['Month', 'Occupation','Credit_Mix','Payment_of_Min_Amount',
'Payment_Behaviour', 'Credit_Score']
label_encoder = LabelEncoder()
for column in toEncode:
    data[column] = label_encoder.fit_transform(data[column])
```

After preprocessing the data, we went from 100,000 instances and 27 features to 74652 instances and 32 features.

Feature Selection: 4 Feature Selection Algorithms will be selected and applied onto the data through WEKA, and one data subset will be selected separately.

Method 1: CorrelationAttributeEval

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
0.08069 10 Delay_from_due_date
0.05262 13 Num_Credit_Inquiries
0.04484 9 Num_of_Loan
0.02833 31 personal_loan
0.0261 15 Outstanding_Debt
0.02551 25 debt_consolidation_loan
0.02287 21 Payment_Behaviour
0.02244 23 auto_loan
0.02057 26 home_equity_loan
0.02003 24 credit_builder_loan
0.01703 27 mortgage_loan
0.01683 30 payday_loan
0.01486 29 not_specified
0.00501 3 Occupation
0.00482 32 student_loan
0.00318 1 Month
0 28 no_loan
-0.00864 16 Credit_Utilization_Ratio
-0.01213 19 Total_EMI_per_month
-0.02063 20 Amount_invested_monthly
-0.03455 5 Monthly_Inhand_Salary
-0.03914 4 Annual_Income
-0.04084 22 Monthly_Balance
-0.04399 2 Age
-0.08273 17 Credit_History_Age

Cut-off: 0.10 => 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

Method 2: ReliefAttributeEval (Ranker):
0.024477 2 Age
0.017906 7 Num_Credit_Card
0.017332 6 Num_Bank_Accounts
0.017219 8 Interest_Rate
0.015441 15 Outstanding_Debt
0.014112 19 Total_EMI_per_month
0.014064 3 Occupation
0.0127 17 Credit_History_Age

0.011524	11 Num_of_Delayed_Payment
0.01107	4 Annual_Income
0.010864	10 Delay_from_due_date
0.009616	13 Num_Credit_Inquiries
0.00737	12 Changed_Credit_Limit
0.006013	9 Num_of_Loan
0.005844	5 Monthly_Inhand_Salary
0.005668	32 student_loan
0.005657	30 payday_loan
0.005379	29 not_specified
0.005245	27 mortgage_loan
0.005222	24 credit_builder_loan
0.00521	23 auto_loan
0.005032	31 personal_loan
0.004947	25 debt_consolidation_loan
0.0048	26 home_equity_loan
0	28 no_loan
-0.000387	22 Monthly_Balance
-0.009254	20 Amount_invested_monthly
-0.018253	16 Credit_Utilization_Ratio
-0.021545	18 Payment_of_Min_Amount
-0.023212	14 Credit_Mix
-0.032999	21 Payment_Behaviour
-0.060802	1 Month

Cut-off: 0.02 => 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

Method 3: ClassifierAttributeEval (Ranker)

0.1879941	4 Annual_Income
0.1812442	15 Outstanding_Debt
0.1583674	19 Total_EMI_per_month
0.1448428	5 Monthly_Inhand_Salary
0.0667689	14 Credit_Mix
0.0327237	8 Interest_Rate
0.0273498	7 Num_Credit_Card
0.0242182	18 Payment_of_Min_Amount
0.0229147	11 Num_of_Delayed_Payment
0.0222949	6 Num_Bank_Accounts

0.0188205	10	Delay_from_due_date
0.0187684	12	Changed_Credit_Limit
0.0048473	13	Num_Credit_Inquiries
0.0016956	2	Age
0.001023	9	Num_of_Loan
0.0004473	1	Month
0.0003747	31	personal_loan
0.000297	25	debt_consolidation_loan
0.0002325	23	auto_loan
0.0002311	17	Credit_History_Age
0.000171	21	Payment_Behaviour
0.0001605	29	not_specified
0.0001526	3	Occupation
0.0001436	27	mortgage_loan
0.000138	26	home_equity_loan
0.0001307	24	credit_builder_loan
0.0000864	30	payday_loan
0.0000121	32	student_loan
0	28	no_loan
-0.1479401	20	Amount_invested_monthly
-0.1575615	22	Monthly_Balance
-0.1602462	16	Credit_Utilization_Ratio

Cutoff: 0.10 => 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

Method 4: CfsSubsetEval (GreedyStepwise)

Algorithm’s Selected Features:
Num_Bank_Accounts
Num_of_Delayed_Payment
Changed_Credit_Limit
Credit_Mix
Payment_of_Min_Amount

Method 5 (Selecting Subset Manually):
'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'

I selected these features from research on the biggest influencers of Credit Score combined with my own intuition.

Models: Using Decision Tree, Random Forest, SVM, Logistic Regression, and Naive Bayes.

For Metrics: Accuracy to evaluate overall performance, ROC-AUCs to see how well the model tells the difference between classes, and TP and FP rates.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

models = {
   "Decision Tree": DecisionTreeClassifier(),
   "Random Forest": RandomForestClassifier(),
   "Logistic Regression": LogisticRegression(max_iter=1000),
   "Naive Bayes": GaussianNB(),
   "SVM": SVC(probability=True)
}

results = {}

for model_name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   y_pred_proba = model.predict_proba(X_test) if hasattr(model, 'predict_proba') else
None

   accuracy = accuracy_score(y_test, y_pred)

   cm = confusion_matrix(y_test, y_pred)

   tpr = {}
   fpr = {}
   for i in range(cm.shape[0]):
       tp = cm[i, i]
       fn = cm[i, :].sum() - tp
       fp = cm[:, i].sum() - tp
       tn = cm.sum() - (tp + fn + fp)

       tpr[i] = tp / (tp + fn) if (tp + fn) != 0 else 0
```

```
        fpr[i] = fp / (fp + tn) if (fp + tn) != 0 else 0

    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr') if y_pred_proba is
not None else 'N/A'

    results[model_name] = {
        'Accuracy': accuracy,
        'Confusion Matrix': cm,
        'True Positive Rate': tpr,
        'False Positive Rate': fpr,
        'ROC-AUC': roc_auc
    }

for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"Accuracy: {metrics['Accuracy']}")
    print(f"Confusion Matrix: \n{metrics['Confusion Matrix']}")
    print(f"True Positive Rate (TPR): {metrics['True Positive Rate']}")
    print(f"False Positive Rate (FPR): {metrics['False Positive Rate']}")
    print(f"ROC-AUC: {metrics['ROC-AUC']}\n")

best_model = max(results, key=lambda x: results[x]['Accuracy'])
print(f"The best model is: {best_model}")
```

4 Results

CorrelationAttributeEval

Model	Accuracy	ROC-AUC	TPR_Class_0	FPR_Class_0	TPR_Class_1	FPR_Class_1	TPR_Class_2	FPR_Class_2
Decision Tree	0.688	0.755	0.624	0.09	0.682	0.149	0.71	0.285
Random Forest	0.738	0.871	0.644	0.07	0.699	0.103	0.787	0.284
Logistic Regression	0.637	0.778	0.365	0.06	0.474	0.096	0.806	0.533
Naive Bayes	0.629	0.769	0.754	0.186	0.612	0.151	0.6	0.236

SVM	0.659	0.793	0.358	0.057	0.498	0.073	0.836	0.527

ReliefAttributeEval

Model	Accuracy	ROC-AUC	TPR_Class_0	FPR_Class_0	TPR_Class_1	FPR_Class_1	TPR_Class_2	FPR_Class_2
Decision Tree	0.539	0.677	0.54	0.148	0.438	0.216	0.593	0.402
Random Forest	0.554	0.709	0.478	0.118	0.39	0.181	0.664	0.477
Logistic Regression	0.528	0.674	0.068	0.046	0.232	0.11	0.825	0.778
Naive Bayes	0.556	0.714	0.135	0.028	0.273	0.112	0.834	0.747
SVM	0.544	0.718	0	0	0	0	1	1

ClassifierAttributeEval

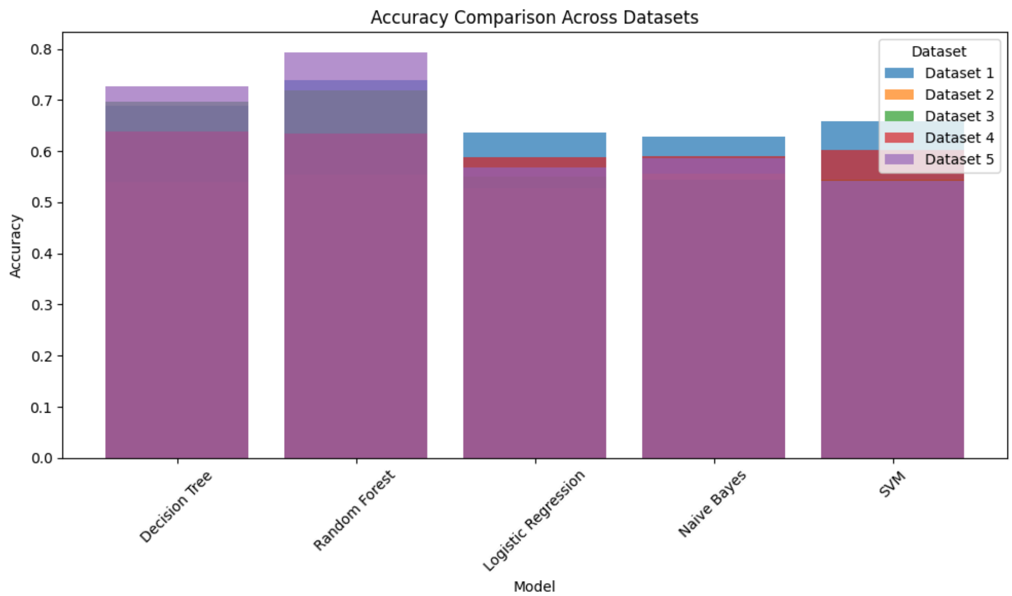
Model	Accuracy	ROC-AUC	TPR_Class_0	FPR_Class_0	TPR_Class_1	FPR_Class_1	TPR_Class_2	FPR_Class_2
Decision Tree	0.697	0.751	0.615	0.083	0.687	0.126	0.726	0.313
Random Forest	0.719	0.861	0.378	0.033	0.721	0.104	0.821	0.391
Logistic Regression	0.551	0.668	0.004	0.001	0.284	0.111	0.859	0.807
Naive Bayes	0.544	0.66	0.233	0.133	0.588	0.18	0.614	0.474
SVM	0.544	0.69	0.006	0.002	0.231	0.098	0.874	0.841

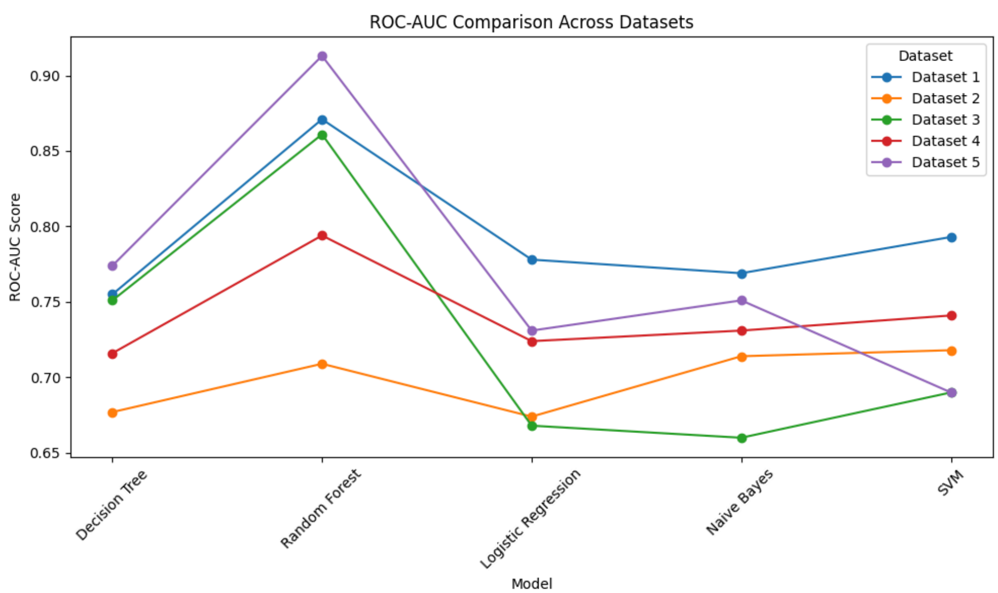
CfsSubsetEval

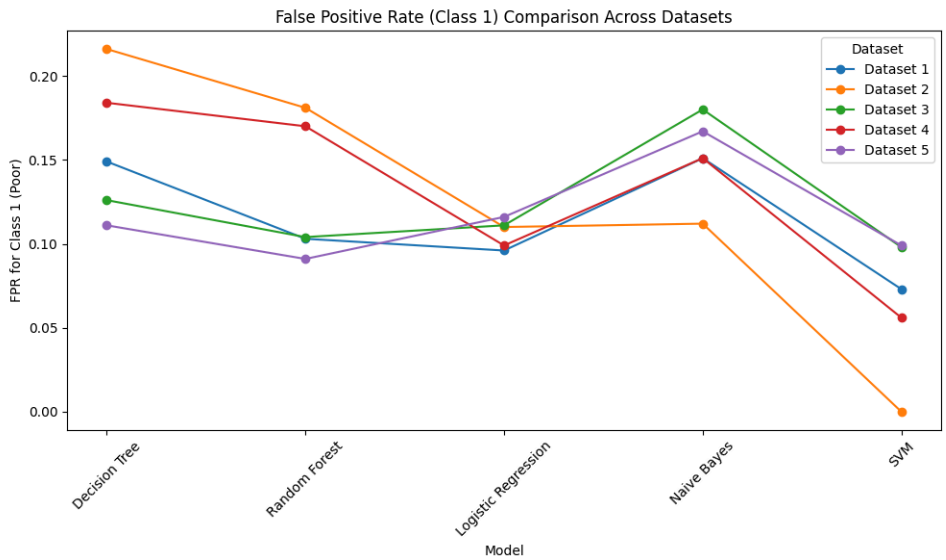
Model	Accuracy	ROC-AUC	TPR_Class_0	FPR_Class_0	TPR_Class_1	FPR_Class_1	TPR_Class_2	FPR_Class_2
Decision Tree	0.638	0.716	0.57	0.101	0.63	0.184	0.662	0.32
Random Forest	0.635	0.794	0.521	0.089	0.577	0.17	0.701	0.37
Logistic Regression	0.588	0.724	0.276	0.066	0.36	0.099	0.805	0.625
Naive Bayes	0.59	0.731	0.662	0.164	0.474	0.151	0.63	0.361
SVM	0.603	0.741	0.386	0.094	0.304	0.056	0.829	0.608

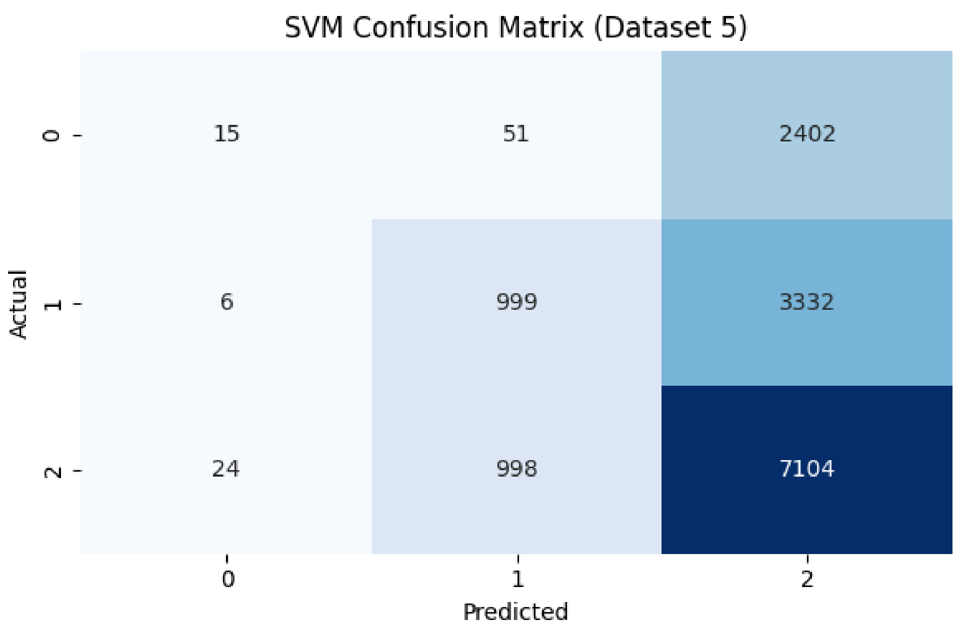
My Feature Selection:

Model	Accuracy	ROC-AUC	TPR_Class_0	FPR_Class_0	TPR_Class_1	FPR_Class_1	TPR_Class_2	FPR_Class_2
Decision Tree	0.727	0.774	0.644	0.074	0.714	0.111	0.759	0.287
Random Forest	0.793	0.913	0.706	0.045	0.801	0.091	0.815	0.228
Logistic Regression	0.569	0.731	0.16	0.04	0.353	0.116	0.809	0.689
Naive Bayes	0.587	0.751	0.757	0.256	0.69	0.167	0.481	0.173
SVM	0.543	0.69	0.006	0.002	0.23	0.099	0.874	0.842


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


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


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


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