CREDIT CARD APPROVAL PREDICTIONS

Varuntej Kodandapuram

I. INTRODUCTION:

Credit risk modeling is one of the most critical applications of analytics and machine learning for financial institutions and banks. The loan approval decision process involves significant credit risk assessment to identify good applicants while preventing defaults.

Banks incur major losses when customers default on credit card debt or other loans. Hence building highly accurate credit risk models is crucial. At the same time, improper applicant rejection also has a negative business impact.

Traditional statistical methods for credit scoring face challenges on complex real-world data. Machine learning has emerged as a powerful approach to improve predictive accuracy. A variety of techniques like logistic regression, neural networks, decision trees and support vector machines have been applied for credit analysis.

This project focuses on building machine learning models to predict credit card approval decisions based on applicant details provided by banks. The machine learning models explored are:

- Logistic Regression
- Decision Trees
- Support Vector Machines (SVM)
- Perceptron Neural Network

These models are trained on a hypothetical dataset that is gathered from the internet and evaluated on classification accuracy, confusion matrix and other metrics.

The report is organized into sections covering the problem definition, algorithm details, experiment setup, results analysis and conclusions. Key academic research on similar credit card approval prediction problems is also reviewed.

This project provides a practical machine learning workflow for an important real-world application. The effectiveness of different modeling approaches is benchmarked. This learning can be extended to other classification tasks as well.

II. MOTIVATION:

The motivation behind this project is to explore different predictive modeling techniques that can accurately classify good and high-risk customers. This project can be used by both the applicants and financial institutions for credit card approval prediction.

The other major goal for this project is to test the accuracies of different modeling algorithms like Logistic Regression, Decision Trees, SVM & Perceptron on real-world credit approval data to determine the most effective modeling algorithm..

Additionally, there are innovation opportunities as traditional models have limitations which latest machine learning advances can address. Finally, extensive academic research interest in this field allows testing on publicly available datasets.

The report analysis would equip financial institutions and credit card applicants with exactly the type of models and predictive capabilities they need in the market.

III. FORMAL STATEMENT OF THE COMPUTATIONAL PROBLEM:

The credit card approval prediction problem can be described as a binary classification problem in machine learning.

❖ **Input Data:**

The input is application records with the following attributes:

- Categorical attributes: gender, education level
- Numerical attributes: age, income, amount, years of employment
- Target attributes: Binary approved or not approved

Gender	Age	Income	Marital_Status	number_of_children	Education_status	Employed

Figure 1: some rows of the data.

❖ **Output Variable:**

The output is a binary value:

- 0 = Credit card application declined
- 1 = Credit card application approved

Approval_Status
1
1

Figure 2: Approval status values.

❖ **Performance Measure:**

- Classification accuracy

Problem Statement:

Given all these attributes involving demographics, income, credit history etc., attempt to predict whether a new credit card applicant will get approved or not based on trained patterns. The objective is to maximize the predictive accuracy across four different machine learning models.

IV. PREVIOUS WORK:

There have been many previous works done in this domain and there have also been many publications addressing this prediction model. One such notable publication is the “Prediction of Credit Card Approval”by Harsha Vardhan Peela, Tanuj Gupta, Nishit Rathod, Tushar Bose, Neha Sharma published in the International Journal of Soft Computing & Engineering. The key computational methodologies that were implemented are:

- **Data Cleaning & Manipulation**: The raw data that they took in as input contained missing values, non-numeric data, and values from different ranges.

  Cleaning & manipulation steps included:

  - Converting non-numeric data to numeric using label encoding
  - Splitting data into train and test sets
  - Scaling feature values to a uniform range using MinMaxScaler

- **Data Analysis:** Performed analysis on feature distributions and correlations to gain insights. Visualizations that are included: heatmaps, pairplots.

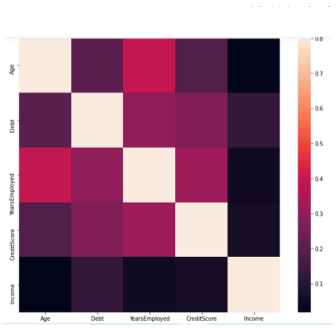

Figure 3: Heatmap of correlation matrix

Figure 4: SNS pairplot

- **Feature Selection**: Selected most useful features and removed less useful ones like Drivers License and Zip Code.
- **Model Building:** They tried two supervised ML models: Logistic Regression and Random Forest Classifier. Evaluation metric used was accuracy score.
- **Model Analysis:** They Analyzed feature importance and model errors to understand about the prediction process. Visualizations included bar charts of education level, gender etc.

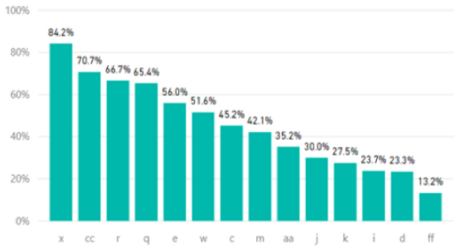

Figure 5: Visualization of education level

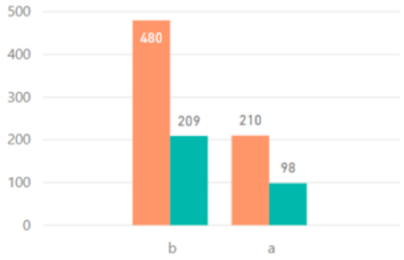

Figure 6: Visualization of gender distribution

In summary, these were the main key computational methodologies that were implemented in their project. When we compare their model to ours there are some quite unique changes such as we are going to compare the ML models

accuracies against each other and determine a model that best suits this type of prediction and we are also implementing four ML models instead of two and also we are also plotting bar graphs that shows us which feature was heavily considered while making the predictions.

V. ALGORITHM DESIGN & SOFTWARE IMPLEMENTATION:

The main software libraries that were used in the project are Pandas for data manipulation, Matplotlib for visualizations, Sklearn for modeling, metrics and data cleaning & manipulation. The coding language that was used is Python and IDE or the editor is Jupyter Notebook. The workflow includes data processing, splitting data, training algorithms, making predictions and model evaluation based on the accuracies.

Data processing:

- Gathered the data from an UCI credit approval dataset and labeled the columns with their respective titles.
- Then, there were certain random characters in certain columns of the data instead of numerical values in order to protect the confidentiality of the applicants. We had to swap them to numerical values to train our models.

Education_status	Employed
q	h
q	h
w	v
w	v
m	v
r	h
cc	v
k	h
w	v
c	h
c	h
k	v
k	v
q	v
k	v
m	v
q	v
d	h
cc	h
c	v
c	v
c	v
x	h
q	v
c	h
i	bb
d	bb
e	h
w	v
aa	v

Education_status	Employed
0	0
0	0
1	1
1	1
2	1
3	0
4	1
5	0
1	1
6	0
6	0
5	1
5	1
0	1
5	1
2	1
0	1
7	0
4	0

Figure 7: Data files before & after manipulation.

❖ Then we built a correlation matrix to see which attributes are crucial for predicting the accuracy, so that we can remove other columns which are not that important.

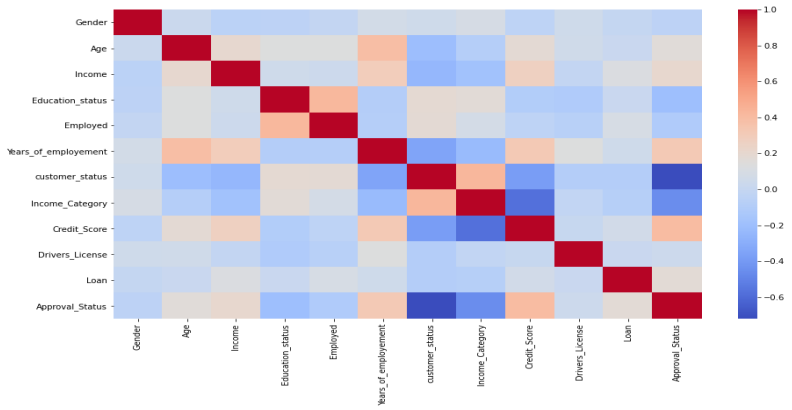

Figure 8: Correlation matrix.

Data Splitting:

- The entire data we are using for our models is in a single csv file.
- We have to train our machine learning models using this data so in order to do the we split the data into two parts:
  - ☐ **Training Data:** This data will be used to train our machine learning models.
  - ☐ **Testing Data:** This data will be used to test our machine learning models for accuracy.

	Gender	Age	Income	Education_status	Employed	Years_of_employement	customer_status
1	0	24.50	0.500	0	0	1.500	0
333	1	34.75	2.500	4	2	0.500	1
506	0	24.75	3.000	0	0	1.835	0
471	1	17.42	6.500	9	1	0.125	1
566	0	25.17	2.875	8	0	0.875	0
...	...	...	...	...	...	...	...
249	1	40.25	21.500	10	5	20.000	0
101	1	18.67	5.000	0	1	0.375	0
131	0	47.42	8.000	10	2	6.500	0
208	1	39.50	4.250	6	2	6.500	0
170	1	41.33	0.000	6	2	15.000	0

447 rows × 11 columns

Figure 9: Training data rows and columns.

	Gender	Age	Income	Education_status	Employed	Years_of_employement	customer_status
349	0	26.17	2.000	13	4	0.000	1
72	1	44.25	0.500	2	1	10.750	0
319	1	21.25	1.500	1	1	1.500	1
209	1	39.33	5.875	4	0	10.000	0
636	1	19.50	9.585	11	1	0.790	1
...	...	...	...	...	...	...	...
331	1	34.08	2.500	6	1	1.000	1
476	1	39.17	2.500	9	0	10.000	1
576	1	25.17	6.000	6	1	1.000	0
659	1	22.25	9.000	11	1	0.085	1
48	1	23.92	0.665	6	1	0.165	1

242 rows × 11 columns

Figure 10: Testing data rows and columns.

The algorithms (ML models) that were implemented are as follows:

➔ **Logistic Regression:** Logistic regression is a statistical method for predicting binary outcomes by fitting data to a logistic function. It calculates the probability of an event occurring using the logistic sigmoid function.

Implementation: sklearn.linear_model LogisticRegression is used & we also plotted a bar graph to show which feature was heavily considered while predicting the accuracy by the Linear regression model using matplotlib.pyplot.

```
# Function to evaluate logistic regression

def evaluate_logistic_regression(train_data, train_decider_column, test_data,
                                 test_decider_column):
    # Logistic Regression Model
    model = LogisticRegression(solver='lbfgs', max_iter=1000)

    # Train the model on the training set
    model.fit(train_data, train_decider_column)

    # Make predictions on the test set
    prediction = model.predict(test_data)

    # Calculate accuracy
    logistic_regression_accuracy = accuracy_score(test_decider_column, prediction)

    print(f"Accuracy for testing set: {logistic_regression_accuracy* 100:.2f}%")
    print("\nClassification Report:\n", classification_report(test_decider_column,
                                                             prediction))
    print("Confusion Matrix:\n", confusion_matrix(test_decider_column, prediction))

    # Extracting feature importances (coefficients)
    feature_importance = np.abs(model.coef_[0])
    feature_names = test_data.columns

    # Creating a bar plot to visualize feature importances
    plt.figure(figsize=(8, 6))
    plt.bar(feature_names, feature_importance)
    plt.title('Feature Importance in Logistic Regression Model')
    plt.xlabel('Feature Names')
    plt.ylabel('Absolute Coefficient Values')
    plt.xticks(rotation=45, ha='right')
    plt.show()

    return logistic_regression_accuracy
```

```
# Evaluate logistic regression
evaluate_logistic_regression(train_data, train_decider_column, test_data,
                             test_decider_column)
```

Figure 11: Implementation of logistic regression.

➔ **Decision Trees:** Decision trees make predictions by learning decision rules to split data based on feature values.

Implementation: Scikit-learn's DecisionTreeClassifier. But in decision trees classification we are trying to find accuracies with different depths in order to find a good fit:

- Under fitting structure: Max depth is 4
- Over fitting structure: Max depth is 10
- Good fit structure: Max depth is 5

& we also plotted a bar graph to show which feature (through sklearn.feature_selection) was heavily considered while predicting the accuracy by the Decision trees model using matplotlib.pyplot.

```
# Function to evaluate decision tree

def evaluate_decision_tree(train_data, train_decider_column, test_data,
                           test_decider_column, max_depth):

    # Initializing and training the Decision Tree model
    decision_trees = DecisionTreeClassifier(max_depth=max_depth)
    decision_trees.fit(train_data, train_decider_column)

    # Predictions and evaluation on the training set
    data_train_prediction = decision_trees.predict(train_data)
    decision_tree_train_accuracy = accuracy_score(data_train_prediction,
                                                  train_decider_column)
    print(f"Accuracy for the Training set: {decision_tree_train_accuracy * 100:.2f}%")
    print("\nClassification Report:\n", classification_report(train_decider_column,
                                                             data_train_prediction))
    print("Confusion Matrix:\n", confusion_matrix(train_decider_column,
                                                  data_train_prediction))

    # Predictions and evaluation on the testing set
    data_test_prediction = decision_trees.predict(test_data)
    decision_tree_test_accuracy = accuracy_score(data_test_prediction,
                                                 test_decider_column)
    print(f"Accuracy for the Testing set: {decision_tree_test_accuracy * 100:.2f}%")
    print("\nClassification Report:\n", classification_report(test_decider_column,
                                                             data_test_prediction))
    print("Confusion Matrix:\n", confusion_matrix(test_decider_column,
                                                  data_test_prediction))

    # Feature Importance
    feature_importance = decision_trees.feature_importances_
    feature_names = test_data.columns

    # Creating a bar plot to visualize feature importances
    plt.figure(figsize=(8, 6))
    plt.bar(feature_names, feature_importance)
    plt.title('Feature Importance in Decision Tree Model')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance Score')
    plt.xticks(rotation=45, ha='right')
    plt.show()


    return decision_tree_test_accuracy
```

```
# with depth 4
evaluate_decision_tree(train_data, train_decider_column, test_data,
                       test_decider_column, max_depth=4)
```

```
# with depth 10
evaluate_decision_tree(train_data, train_decider_column, test_data,
                       test_decider_column, max_depth=10)
```

```
# with depth 5
evaluate_decision_tree(train_data, train_decider_column, test_data,
                       test_decider_column, max_depth=5)
```

Figure 12: Implementation of decision trees.

➔ **SVM (Support Vector Machine) Classifier:** SVM tries to find the hyperplane with maximum margin to separate classes. Has capability to do non-linear separation using kernel.

Implementation: sklearn.svm used.In this model we are calculating accuracies in two different kernels:

- Kernel 1: linear
- Kernel 2: rbf

& we also plotted a bar graph to show which feature (through sklearn.feature_selection), was heavily considered while predicting the accuracy by the SVM classifier using matplotlib.pyplot.

```
# Function to evaluate SVM
def evaluate_svm(train_data, test_data, train_decider_column, test_decider_column,
                 n_components, C, kernel):


    # Initializing and fitting SVM model

    my_model = SVC(C=C, kernel=kernel)
    my_model.fit(train_data, train_decider_column)

    # Predictions and evaluation

    prediction_decider_column = my_model.predict(test_data)
    svm_accuracy = accuracy_score(test_decider_column, prediction_decider_column)

    print(f"Accuracy for the Testing set: {svm_accuracy * 100:.2f}%")
    print(f"Classification Report (C={C}, kernel='{kernel}'):")
    print(classification_report(test_decider_column, prediction_decider_column))
    print("Confusion Matrix:")
    print(confusion_matrix(test_decider_column, prediction_decider_column))
    print("\n")

    if kernel == 'linear':
        feature_importance = np.abs(my_model.coef_.flatten())
        feature_names = train_data.columns

        # Creating a bar plot to visualize feature importances
        plt.figure(figsize=(12, 6))
        plt.bar(feature_names, feature_importance)
        plt.title('Feature Importance in Linear SVM Model')
        plt.xlabel('Feature Names')
        plt.ylabel('Coefficient Magnitude')
        plt.xticks(rotation=45, ha='right')
        plt.show()

    elif kernel == 'rbf':
        result = permutation_importance(my_model, test_data, test_decider_column,
                                        n_repeats=10, random_state=42)
        feature_importance = result.importances_mean
        feature_names = test_data.columns

        # Creating a bar plot to visualize feature importances
        plt.figure(figsize=(8, 6))
        plt.bar(feature_names, feature_importance)
        plt.title('Permutation Importance in rbf SVM Model')
        plt.xlabel('Feature Names')
        plt.ylabel('Permutation Importance')
        plt.xticks(rotation=45, ha='right')
        plt.show()

    return svm_accuracy
```

```
# Evaluating SVM with different parameters

evaluate_svm(train_data, test_data, train_decider_column, test_decider_column,
             n_components=4, C=0.1, kernel='linear')
```

```
evaluate_svm(train_data, test_data, train_decider_column, test_decider_column,
             n_components=4, C=0.1, kernel='rbf')
```

Figure 13: Implementation of SVM classifier.

➔ **Perceptron:** Perceptron is a linear binary classifier that separates data points with a straight line. Errors are used to update weights.

Implementation: sklearn.linear_model Perceptron model used. We also plotted a bar graph to show which feature (through sklearn.feature_selection) was heavily considered while predicting the accuracy by the Perceptron model using matplotlib.pyplot.

```
# Function to train and evaluate perceptron
def evaluate_perceptron(train_vectors, train_labels, test_vectors, test_labels):
    # Initialising a Perceptron
    percep = Perceptron()
    # Fit the model to the training data
    percep.fit(train_data, train_decider_column)
    # Make predictions on the test data
    prediction_percep = percep.predict(test_data)
    perceptron_accuracy = accuracy_score(test_decider_column, prediction_percep)
    # Print the accuracy score
    print(f"Accuracy for the Testing set: {perceptron_accuracy * 100:.2f}%")

    # Print the classification report
    print("Classification report:")
    print(classification_report(test_decider_column, prediction_percep))
    # Print the confusion matrix
    print('Confusion matrix:')
    print(confusion_matrix(test_decider_column, prediction_percep))
    # Feature Importance
    feature_importance = np.abs(percep.coef_.flatten())
    feature_names = test_data.columns
    # Create a bar plot to visualize feature importances
    plt.figure(figsize=(8, 6))
    plt.bar(feature_names, feature_importance)
    plt.title('Feature Importance in Perceptron Model')
    plt.xlabel('Feature Names')
    plt.ylabel('Coefficient Magnitude')
    plt.xticks(rotation=45, ha='right')
    plt.show()

    # Return the perceptron accuracy
    return perceptron_accuracy

# Evaluate the perceptron model
evaluate_perceptron(train_data, train_decider_column, test_data, test_decider_column)
```

Figure 14: Implementation of Perceptron.

VI. RESULTS & ANALYSIS:

We have already seen & discussed the algorithm design and implementation. Now let's look at the accuracies generated by the four ML models:

➔ **Logistic Regression:**

```
Accuracy for testing set: 88.02%

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88       128
           1       0.85      0.91      0.88       114

    accuracy                           0.88       242
   macro avg       0.88      0.88      0.88       242
weighted avg       0.88      0.88      0.88       242

Confusion Matrix:
 [[109  19]
 [ 10 104]]
```

Figure 15: Results of logistic regression. model.

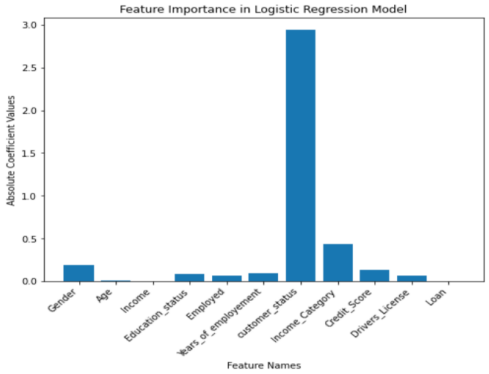

Figure 16: Feature importance in a logistic regression model.

Here we can see that this model predicts credit card approval with an accuracy of 88.02% for the test data that we have provided. There is also a classification report and a confusion matrix provided along with the accuracy score in fig 15. Based on the bar graph visualization i.e. fig 16 we can state that the customer status feature is heavily considered while calculating making the predictions.

➔ **Decision Trees:**

Max depth 4:

```
Accuracy for the Testing set: 83.88%

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       128
           1       0.89      0.75      0.82       114

    accuracy                           0.84       242
   macro avg       0.85      0.83      0.84       242
weighted avg       0.84      0.84      0.84       242

Confusion Matrix:
 [[117  11]
 [ 28  86]]
```

Figure 17: Accuracy of decision trees.

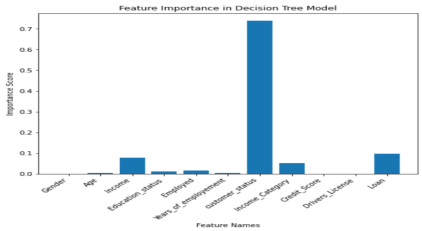

Figure 18: Feature importance in Decision Trees.

Max depth 10:

```
Accuracy for the Testing set: 81.40%

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       128
           1       0.79      0.82      0.81       114

    accuracy                           0.81       242
   macro avg       0.81      0.81      0.81       242
weighted avg       0.81      0.81      0.81       242

Confusion Matrix:
 [[104  24]
 [ 21  93]]
```

Figure 19: Accuracy of decision Trees.

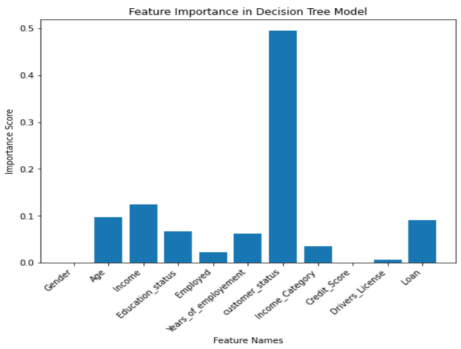

Figure 20: Feature importance in Decision Trees.

Max depth 5:

```
Accuracy for the Testing set: 86.78%

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       128
           1       0.84      0.89      0.86       114

    accuracy                           0.87       242
   macro avg       0.87      0.87      0.87       242
weighted avg       0.87      0.87      0.87       242

Confusion Matrix:
 [[109  19]
 [ 13 101]]
```

Figure 21: Accuracy of decision Trees.

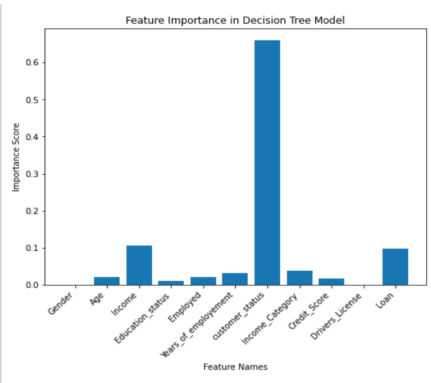

Figure 22: Feature importance in Decision Trees.

Here , for decision trees we calculate accuracies for three different structures i.e fig 17,19, 21 and we also show a bar graph visualization to determine which feature was heavily considered while making the predictions as you can see in fig 18, 20, 22:

1. This underfitting structure of max depth = 4 has an accuracy of 83.88%.

   Important feature: customer status.

2. This overfitting structure of max depth = 10 has an accuracy of 81.40%.

   Important feature: Customer status.

3. This goodfit structure of max depth = 5 has an accuracy of 86.78%.

   Important feature: Customer status.

Therefore the max depth = 5 has provided us a better accuracy in predictions when compared to other depths.

➔ **SVM Classifier:**

Kernel: linear

```
Accuracy for the Testing set: 87.19%
Classification Report (C=0.1, kernel='linear'):
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       128
           1       0.82      0.93      0.87       114

    accuracy                           0.87       242
   macro avg       0.88      0.88      0.87       242
weighted avg       0.88      0.87      0.87       242

Confusion Matrix:
[[105  23]
 [  8 106]]
```

Figure 23: Accuracy of SVM Classifier.

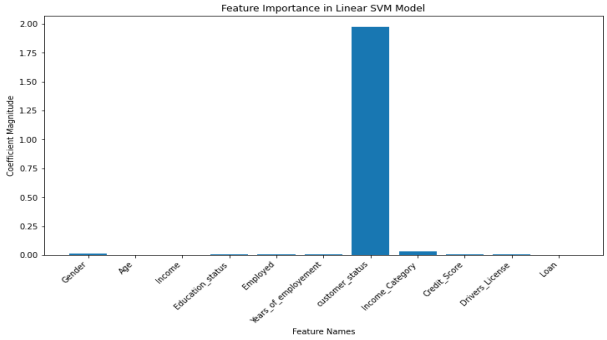

Figure 24: Feature importance in SVM classifier.

Kernel: rbf

```
Accuracy for the Testing set: 57.85%
Classification Report (C=0.1, kernel='rbf'):
              precision    recall  f1-score   support

           0       0.56      0.98      0.71       128
           1       0.88      0.12      0.22       114

    accuracy                           0.58       242
   macro avg       0.72      0.55      0.46       242
weighted avg       0.71      0.58      0.48       242

Confusion Matrix:
[[126   2]
 [100  14]]
```

Figure 25: Accuracy of SVM Classifier.

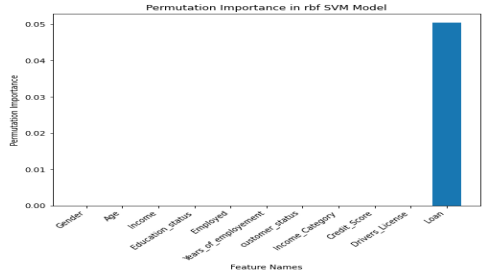

Figure 26: Feature importance in SVM classifier.

Here, for the SVM classifier we are calculating accuracies in two kernels i.e. fig 23, 25,we also show a bar graph visualization to determine which feature was heavily considered while making the predictions as you can see in fig 24, 26:

1. The “linear” kernel gives us an accuracy of 87.19%. Important feature: customer status.

2. The “rbf” kernel gives us an accuracy of 57.85%. Important feature: Loan.

The linear kernel has best accuracy because the attributes have a linear relationship with the target attribute (credit card approval).

➔ **Perceptron:**

```
Accuracy for the Testing set: 61.16%
Classification report:
              precision    recall  f1-score   support

           0       0.68      0.56      0.61       133
           1       0.56      0.68      0.61       109

    accuracy                           0.61       242
   macro avg       0.62      0.62      0.61       242
weighted avg       0.62      0.61      0.61       242

Confusion matrix:
[[74 59]
 [35 74]]
```

Figure 27: Accuracy of Perceptron.

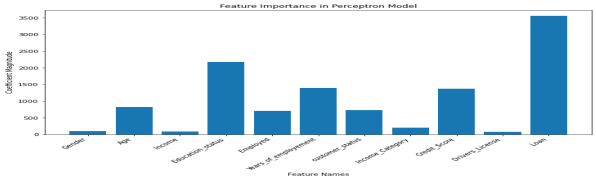

Figure 28: Feature important in Perceptron.

Here we can see that this model predicts credit card approval with an accuracy of 61.16% as you can see in fig 27, for the test data that we have provided. There is also a classification report and a confusion matrix provided along with the accuracy score. This is the case because unlike SVM kernels or decision trees using different depths, perceptron doesn’t perform feature engineering.Based on the bar graph visualization i.e. fig 28 we can state that the loan feature is heavily considered while making the predictions. Now, let’s compare all the four model

accuracies using a bar graph & a line graph to see which model is giving the best accuracy in predicting the credit card approval.

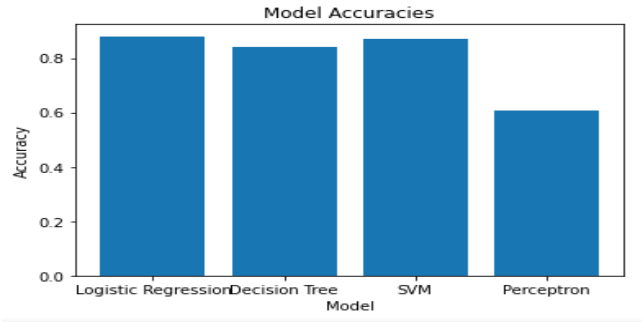

Figure 29: Bar graph comparing all the the four models

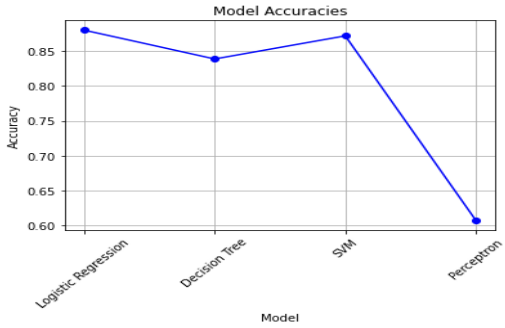

Figure 30: Line graph comparing all the four models

VII. CONCLUSIONS:

The credit card approval rate prediction model demonstrates promising results. Based on the visualizations, we can clearly state that the logistic regression model has the highest accuracy of 88.02%. When compared to all four models, this is mainly because the relationship between the features and target variables is largely linear in nature. Logistic regression excels at modeling linear decision boundaries. We know that even an SVM classifier with a linear kernel is also good at handling linear decision boundaries. That is the reason these two models differ by just 0.83% in accuracy, giving the logistic regression model an upper hand in predictions.

By the visualizations of feature importance graphs in the results and analysis section, we can state that the customer status feature was the most important element while predicting accuracy. It's not the case that other features were not considered—every feature was considered while making the predictions, but customer status played a huge role. We can also see that models that considered the loan feature as important had

lower accuracy scores when compared to other models which considered customer status.

In conclusion, applicants and financial institutions can adapt the Logistic Regression Model for credit card approval predictions.

VIII. SCOPE FOR FUTURE WORK:

There are certain areas where further work can be done in order to improve the model like:

A. How can the models be improved to increase accuracy further?

   The highest accuracy achieved was around 80-85%. There may be opportunities to tune model hyperparameters, add additional features, or try different algorithms like neural networks to improve performance.

B. How can the models account for changes in applicant profiles and economic conditions over time?

   Updating the models continuously through re-training as new data arrives could maintain predictive power as underlying data distributions shift.

C. How well do these models generalize to new unseen data?

   The models were only trained and tested on a single dataset. Evaluating performance on entirely new datasets from different time periods or populations would better assess real-world usefulness.

Future research can be done in these areas and enhanced models could be implemented.

IX. REFERENCES:

Credit Card Approval Prediction Publication:

https://www.ijsce.org/wp-content/uploads/papers/v11i2/B35350111222.pdf

Dataset:

https://archive.ics.uci.edu/dataset/27/credit+approval

Linear Regression:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Decision Tree Classification:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Support Vector Machine (SVM) Classifier:

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Perceptron:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html