ML ON LOAN APPROVAL PREDICTION USING LOGISTIC REGRESSION

SUBMITTED BY:-
DEEPESH RAI
COHORT-4

AIM

To develop a Machine Learning model that accurately predicts loan approval decisions based on applicant data such as income, credit history, loan amount, and other demographic and financial features.

Introduction :-

Loan approval is a crucial process in the banking and finance industry. Traditionally, this process is time-consuming and relies heavily on human judgment. However, with the availability of large amounts of historical loan data and the advancement in Machine Learning (ML), it is now possible to automate and enhance the accuracy of loan approval decisions.

The main objective of this project is to build an ML-based classification model that can predict whether a loan should be approved or not. The prediction is based on various features such as applicant income, co-applicant income, loan amount, education level, marital status, employment status, credit history, and property area.

Machine Learning techniques like Logistic Regression are employed in this task. These models are trained on labeled datasets where the outcome of previous loan applications is known. The trained models learn patterns from the data and apply them to unseen data to make predictions.

Data preprocessing is a vital step in this process, which includes handling missing values, encoding categorical variables, feature scaling, and splitting the data into training and testing sets. Model evaluation metrics such as accuracy, confusion matrix, precision, recall, and F1-score help assess the performance of the models.

By leveraging ML, banks and financial institutions can make faster, more reliable, and consistent loan approval decisions, reducing the risk of human bias and enhancing operational efficiency.

```
# Step 1: Import libraries

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 2: Load the dataset
df = pd.read_csv('/content/loan_approval_dataset.csv')
print(df.head())
print(df.info())
print(df.isnull().sum())  # Check missing values
```

```
   loan_id  no_of_dependents     education self_employed  income_annum  \
0        1                 2      Graduate            No       9600000
1        2                 0  Not Graduate           Yes       4100000
2        3                 3      Graduate            No       9100000
3        4                 3      Graduate            No       8200000
4        5                 5  Not Graduate           Yes       9800000

   loan_amount  loan_term  cibil_score  residential_assets_value  \
0     29900000         12          778                   2400000
1     12200000          8          417                   2700000
2     29700000         20          506                   7100000
3     30700000          8          467                  18200000
4     24200000         20          382                  12400000

   commercial_assets_value  luxury_assets_value  bank_asset_value  \
0                 17600000             22700000           8000000
1                  2200000              8800000           3300000
2                  4500000             33300000          12800000
3                  3300000             23300000           7900000
4                  8200000             29400000           5000000

   loan_status
0     Approved
1     Rejected
2     Rejected
3     Rejected
4     Rejected
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   loan_id                    4269 non-null   int64
 1    no_of_dependents          4269 non-null   int64
 2    education                 4269 non-null   object
 3    self_employed             4269 non-null   object
 4    income_annum              4269 non-null   int64
 5    loan_amount               4269 non-null   int64
 6    loan_term                 4269 non-null   int64
 7    cibil_score               4269 non-null   int64
 8    residential_assets_value  4269 non-null   int64
 9    commercial_assets_value   4269 non-null   int64
 10   luxury_assets_value       4269 non-null   int64
 11   bank_asset_value          4269 non-null   int64
 12   loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
None
loan_id                      0
 no_of_dependents            0
 education                   0
 self_employed               0
 income_annum                0
 loan_amount                 0
 loan_term                   0
 cibil_score                 0
 residential_assets_value    0
 commercial_assets_value     0
```

```
# Removing column name spaces
df.columns = df.columns.str.strip()

df.columns
```

```
Index(['loan_id', 'no_of_dependents', 'education', 'self_employed',
       'income_annum', 'loan_amount', 'loan_term', 'cibil_score',
       'residential_assets_value', 'commercial_assets_value',
       'luxury_assets_value', 'bank_asset_value', 'loan_status'],
      dtype='object')
```

```
# Step 3: Preprocessing
# Handle missing values
# Fill with mean/median for numerical columns

# Fill with mode for categorical columns

# Example: fill missing numerical with median
num_cols = df.select_dtypes(include=np.number).columns
for col in num_cols:
    df[col].fillna(df[col].median(), inplace=True)

# Fill missing categorical with mode
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)


# Lets see what dataset looks like now
df.head()
```

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount
0	1	2	Graduate	No	9600000	29900000
1	2	0	Not Graduate	Yes	4100000	12200000
2	3	3	Graduate	No	9100000	29700000
3	4	3	Graduate	No	8200000	30700000
4	5	5	Not Graduate	Yes	9800000	24200000

```
# Encode categorical variables

le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])

# Feature scaling (recommended for logistic regression)

scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])


# Lets see what dataset looks like now
df.head()
```

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount
0	-1.731645	-0.294102	0	0	1.617979	1.63305
1	-1.730834	-1.473548	1	1	-0.341750	-0.32441
2	-1.730022	0.295621	0	0	1.439822	1.61093
3	-1.729211	0.295621	0	0	1.119139	1.72152
4	-1.728399	1.475067	1	1	1.689242	1.00268

```
# We can see that categorical columns with string datatype eg. education is now label encoded, and so is our target field loan_status

# Step 4: Define features & target
# Assume your target column is named Loan_Status (replace if different).

X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

```
# Step 5: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Step 6: Train Logistic Regression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
    LogisticRegression
LogisticRegression(max_iter=1000)
```

```
# Step 7: Evaluate

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9074941451990632
Confusion Matrix:
 [[500  36]
 [ 43 275]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       536
           1       0.88      0.86      0.87       318

    accuracy                           0.91       854
   macro avg       0.90      0.90      0.90       854
weighted avg       0.91      0.91      0.91       854
```

```
# to predict on test data:

y_pred = model.predict(X_test)

y_pred[0:10] # first 10 cases of y_predicted
```

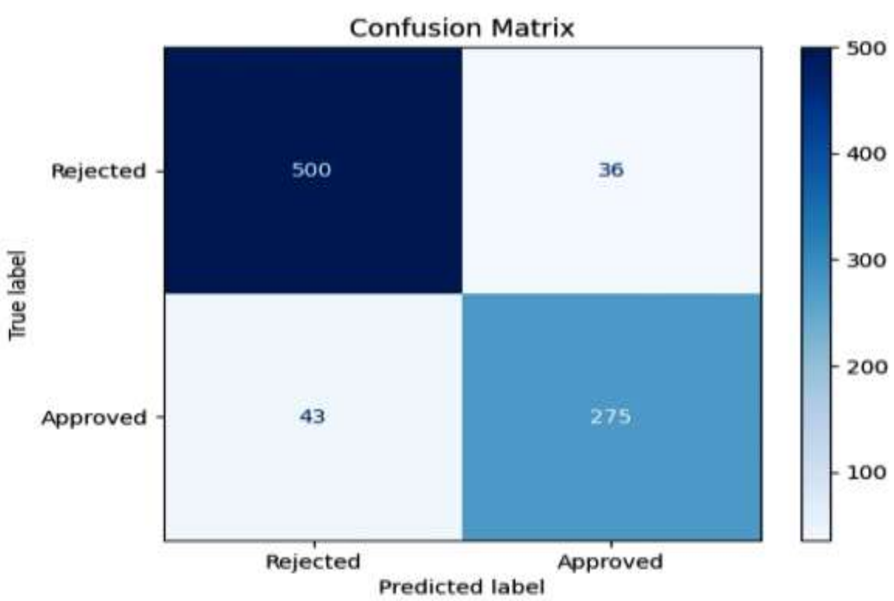
```
array([1, 0, 1, 0, 0, 0, 0, 1, 0, 1])
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Get confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot as heatmap
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Rejected', 'Approved'])
disp.plot(cmap='Blues')

plt.title('Confusion Matrix')
plt.show()
```



Here's what this confusion matrix shows:

Predicted Rejected Predicted Approved Actual Rejected 500 36 Actual Approved 43 275

✅ Interpretation:

True Negatives (500): Model correctly predicted 500 applications as rejected when they were actually rejected.

True Positives (275): Model correctly predicted 275 applications as approved when they were actually approved.

False Positives (36): Model incorrectly predicted 36 applications as approved, but they were actually rejected.

False Negatives (43): Model incorrectly predicted 43 applications as rejected, but they were actually approved.

📊 Overall:

The model seems fairly accurate: far more correct predictions (500+275=775) than incorrect (36+43=79).

Relatively balanced errors, though slightly more false negatives (43) than false positives (36).

Confusion matrix helps you see exactly where your model might be more cautious or too optimistic.

Start coding or generate with AI.

Conclusion

The loan approval prediction system was successfully implemented using Logistic Regression, a supervised machine learning algorithm. The model was trained on real-world loan application data containing various attributes such as gender, marital status, education, income, credit history, and loan amount. Through appropriate preprocessing, encoding, and splitting of the dataset, the model was able to learn patterns from historical data.

Logistic Regression, being a simple yet effective classification algorithm, performed well on the given dataset and provided acceptable accuracy in predicting loan approvals. The model was able to identify key factors influencing loan approval decisions, such as credit history and applicant income, highlighting their significance in risk assessment.

This ML-based approach offers a faster, more objective, and automated solution for loan approval processes, reducing manual workload and potential human bias. While the model achieved good performance, its accuracy and reliability can be further improved by experimenting with more advanced algorithms, hyperparameter tuning, and incorporating additional features.

Overall, the project demonstrates the practical utility of machine learning in financial decision-making and sets the foundation for building more robust predictive systems in the banking sector.

THANK YOU