

Credit Card Default Prediction

L.Ajith kumar, *graduate Student*, Lawrence tech University, Fall 2024

INTRODUCTION & DOMAIN KNOWLEDGE

Provide an overview of the credit card default prediction problem, including its impact on financial institutions and customers. Briefly describe the significance of predicting credit default and highlight the attributes of the dataset (e.g., credit limit, payment status, bill amounts) from the UCI repository. Explain why features such as payment status and bill amounts are relevant.

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000	2		2	1	24	2	-1	-1
1	2	120000	2		2	2	26	-1	2	0
2	3	90000	2		2	2	34	0	0	0
3	4	50000	2		2	1	37	0	0	0
4	5	50000	1		2	1	57	-1	0	-1

	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3
0	0	0	0	0	689	0
1	3272	3455	3261	0	1000	1000
2	14331	14948	15549	1518	1500	1000
3	28314	28959	29547	2000	2019	1200
4	20940	19146	19131	2000	36681	10000

	PAY_AMT4	PAY_AMT5	PAY_AMT6	default	payment	next	month
0	0	0	0				1
1	1000	0	2000				1
2	1000	1000	5000				0
3	1100	1069	1000				0
4	9000	689	679				0

[5 rows x 25 columns]
<class 'pandas.core.frame.DataFrame'
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):

#	Column	Non-Null Count	Dtype
0	ID	30000 non-null	int64
1	LIMIT_BAL	30000 non-null	int64
2	SEX	30000 non-null	int64
3	EDUCATION	30000 non-null	int64
4	MARRIAGE	30000 non-null	int64
5	AGE	30000 non-null	int64
6	PAY_0	30000 non-null	int64
7	PAY_2	30000 non-null	int64
8	PAY_3	30000 non-null	int64
9	PAY_4	30000 non-null	int64
10	PAY_5	30000 non-null	int64
11	PAY_6	30000 non-null	int64
12	BILL_AMT1	30000 non-null	int64
13	BILL_AMT2	30000 non-null	int64
14	BILL_AMT3	30000 non-null	int64
15	BILL_AMT4	30000 non-null	int64
16	BILL_AMT5	30000 non-null	int64

STEP 1 Output

STEP 1

DATASET ANALYSIS & UNDERSTANDING

```
import pandas as pd
data = pd.read_csv('path_to_dataset.csv')
data = pd.read_excel(url, header=1)
# Display the first five rows and dataset info
print(data.head())
print(data.info())
```

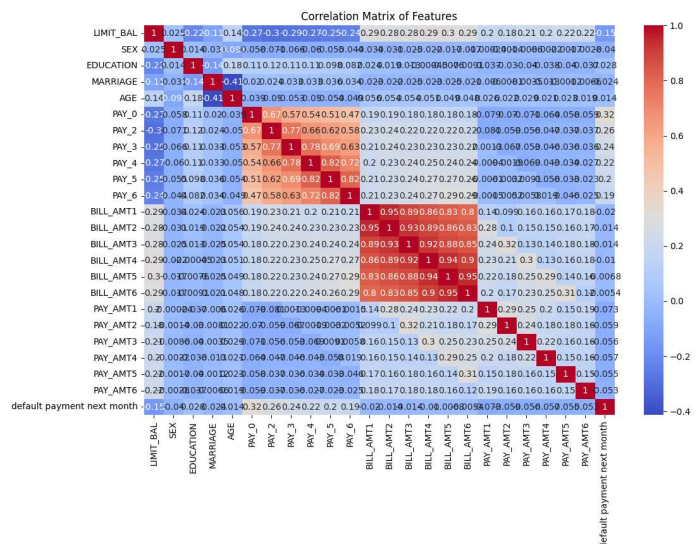
The dataset is loaded, and basic info like column names, data types, and statistical summaries (mean, min, max) for numerical features is reviewed.

STEP 2

Feature Analysis & Selection

```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = data.corr()
plt.figure(figsize=(12,10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
cmap="coolwarm")
plt.title("Correlation Matrix of Features")
plt.show()
```

A correlation heatmap to visualize relationships between features, aiding in feature selection by showing dependencies.



STEP-2Output

STEP 3

Data Cleaning/Preprocessing

```
print("Duplicates:", data.duplicated().sum())
print("Missing values:\n", data.isnull().sum())
# Dropping duplicates (if any)
data = data.drop_duplicates()
```

Identify and handle missing values and duplicates to ensure data consistency.

Duplicates:	0
Missing values:	
ID	0
LIMIT_BAL	0
SEX	0
EDUCATION	0
MARRIAGE	0
AGE	0
PAY_0	0
PAY_2	0
PAY_3	0
PAY_4	0
PAY_5	0
PAY_6	0
BILL_AMT1	0
BILL_AMT2	0
BILL_AMT3	0
BILL_AMT4	0
BILL_AMT5	0
BILL_AMT6	0
PAY_AMT1	0
PAY_AMT2	0
PAY_AMT3	0
PAY_AMT4	0
PAY_AMT5	0
PAY_AMT6	0
default payment next month	0
dtype:	int64

STEP 3 Output

STEP 4

Data Visualization – Independent Features

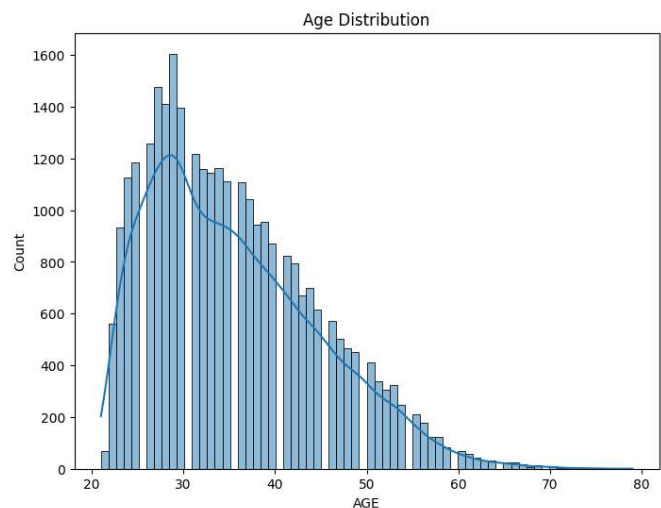
```
# Plotting feature distributions
sns.histplot(data['AGE'], bins=30, kde=True)
plt.title("Age Distribution")
plt.show()

sns.countplot(x='default.payment.next.month', data=data)
plt.title("Default Payment Distribution")
plt.show()
```

STEP 4 EXPLANATION

Plots to understand distributions of individual features, e.g., age and default rates.

```
# Plotting feature distributions
sns.histplot(data['AGE'], bins=30, kde=True)
plt.title("Age Distribution")
plt.show()
sns.countplot(x='default.payment.next.month', data=data)
plt.title("Default Payment Distribution")
plt.show()
```



STEP 4 Output

STEP 5

DATA TRANSFORMATION & MODELS USED

Feature Scaling

from sklearn.preprocessing import StandardScaler

```
scaler = StandardScaler()
numerical_features = ['LIMIT_BAL', 'AGE', 'BILL_AMT1',
                      'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',
                      'BILL_AMT6']
data[numerical_features] =
scaler.fit_transform(data[numerical_features])
```

Standardizes numerical features, improving model performance.

One-hot Encoding for Logistic Regression Classifier

```
data_encoded = pd.get_dummies(data, columns=['SEX',
'EDUCATION', 'MARRIAGE'], drop_first=True)
```

Explanation: Binary encoding for categorical features, creating a more suitable format for Logistic Regression.

Integer Label Encoding for Random Forest Classifier

```
from sklearn.preprocessing import LabelEncoder
data['SEX'] = LabelEncoder().fit_transform(data['SEX'])
data['EDUCATION'] =
LabelEncoder().fit_transform(data['EDUCATION'])
data['MARRIAGE'] =
LabelEncoder().fit_transform(data['MARRIAGE'])
```

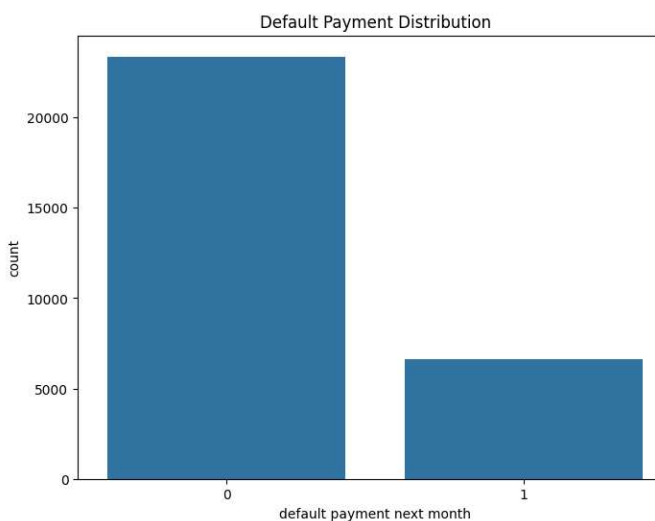
Label encoding for categorical features suitable for Random Forest.

STEP 6

Logistic Regression Classifier

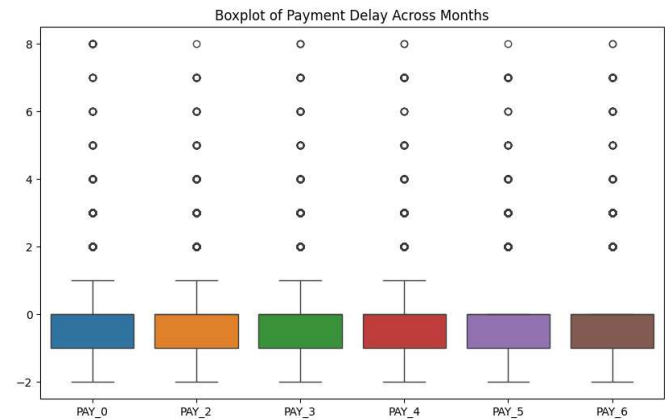
```
from sklearn.linear_model
import LogisticRegression
model_lr = LogisticRegression(class_weight='balanced',
random_state=42)
model_lr.fit(X_train, y_train)
```

Logistic Regression model with balanced class weights to manage class imbalance.



Handling the Data Imbalance

Explanation: Both models utilize `class_weight='balanced'`, addressing class imbalance by weighting the minority class higher.



STEP 7

EXPERIMENTS & MODEL RESULTS

Logistic Regression Tuning & Evaluation

```
from sklearn.model_selection import cross_val_score
import numpy as np
cv_scores_lr = cross_val_score(model_lr, X_train, y_train,
cv=5, scoring='accuracy')
print(f"Logistic Regression CV Accuracy:
Mean={cv_scores_lr.mean():.3f},
Std={cv_scores_lr.std():.3f}")
```

Cross-validation accuracy and standard deviation for Logistic Regression.

Random Forest Tuning & Evaluation

```
cv_scores_rf = cross_val_score(model_rf, X_train, y_train,
cv=5, scoring='accuracy')
print(f"Random Forest CV Accuracy:
Mean={cv_scores_rf.mean():.3f},
Std={cv_scores_rf.std():.3f}")
```

Cross-validation accuracy and standard deviation for Random Forest.

STEP 7 Output:

Mean CV accuracy: 0.81, Standard deviation: 0.00
Test set accuracy: 0.81

STEP 8

```

import gradio as gr

# Prediction function for Gradio app
# Set up Gradio input components with descriptive labels for
payment statuses without numbers
input_features = [
    gr.Slider(minimum=10000, maximum=1000000,
value=50000, label="LIMIT_BAL"),
    gr.Dropdown(choices=["Male", "Female"], label="SEX"),
    gr.Dropdown(choices=["Graduate School", "University",
"High School", "Others"], label="EDUCATION"),
    gr.Dropdown(choices=["Single", "Married", "Divorced",
"Widowed"], label="MARRIAGE"),
    gr.Slider(minimum=21, maximum=79, value=30,
label="AGE"),
    gr.Dropdown(choices=[
        "No consumption", "Paid in full", "Paid minimum due",
        "1 month late", "2 months late", "3 months late",
        "4 months late", "5 months late", "6 months late",
        "7 months late", "8 months late"
    ], label="PAY_1"),
    gr.Dropdown(choices=[
        "No consumption", "Paid in full", "Paid minimum due",
        "1 month late", "2 months late", "3 months late",
        "4 months late", "5 months late", "6 months late",
        "7 months late", "8 months late"
    ], label="PAY_2"),
    gr.Dropdown(choices=[
        "No consumption", "Paid in full", "Paid minimum due",
        "1 month late", "2 months late", "3 months late",
        "4 months late", "5 months late", "6 months late",
        "7 months late", "8 months late"
    ], label="PAY_3"),
    gr.Dropdown(choices=[
        "No consumption", "Paid in full", "Paid minimum due",
        "1 month late", "2 months late", "3 months late",
        "4 months late", "5 months late", "6 months late",
        "7 months late", "8 months late"
    ], label="PAY_4"),
    gr.Dropdown(choices=[
        "No consumption", "Paid in full", "Paid minimum due",
        "1 month late", "2 months late", "3 months late",
        "4 months late", "5 months late", "6 months late",
        "7 months late", "8 months late"
    ], label="PAY_5"),

    gr.Slider(minimum=-165580, maximum=964511,

```

```

value=399465.5, label="BILL_AMT1"),
    gr.Slider(minimum=-69777, maximum=983931,
value=457077, label="BILL_AMT2"),
    gr.Slider(minimum=-157264, maximum=1664089,
value=753412.5, label="BILL_AMT3"),
    gr.Slider(minimum=-170000, maximum=891586,
value=360793, label="BILL_AMT4"),
    gr.Slider(minimum=-81334, maximum=927171,
value=422918.5, label="BILL_AMT5"),

    gr.Slider(minimum=0, maximum=873552, value=436776,
label="PAY_AMT1"),
    gr.Slider(minimum=0, maximum=1684259,
value=842129.5, label="PAY_AMT2"),
    gr.Slider(minimum=0, maximum=896040, value=448020,
label="PAY_AMT3"),
    gr.Slider(minimum=0, maximum=621000, value=310500,
label="PAY_AMT4"),
    gr.Slider(minimum=0, maximum=426529,
value=213264.5, label="PAY_AMT5"),

]

# Create and launch the Gradio interface
gr.Interface(fn=predict_default, inputs=input_features,
outputs="text").launch()

```

Implementing the Gradio App

In a similar manner as above, it was identified that the categorical data would need to undergo some form of feature transformation in order to be converted to a form which could be used for training within a random forest classifier. Because of the ability for tree classifiers to use discrete integer values as criteria in determining node splits, it was determined that an integer label encoding scheme would be optimal for training and classification, and would subsequently result in a significantly lower encoding dimensionality space as compared to the one-hot encoding scheme. Thus, an integer encoding scheme was implemented according to the steps outline in [9] and used as the input for the random forest classifier.

```

# Install Gradio if not already installed
!pip install gradio

```

```

# Import necessary libraries
import gradio as gr
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression #
Example model

```

```

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00350/default%20of%20credit%20card%20client
s.xls"

```

```

data = pd.read_excel(url, header=1)

# Print column names to verify
print("Column Names in Dataset:")
print(data.columns)

# Assuming the target column is 'default payment next month'
target_column = 'default payment next month' # Adjust if necessary

# Features and target
X = data.drop(columns=[target_column])
y = data[target_column]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a model (example using Logistic Regression)
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Function to make predictions
def predict_default(*features):
    try:
        # Scale the input features
        features_array = np.array([features]) # Convert the
input features to a 2D array
        features_scaled = scaler.transform(features_array)

        # Make prediction
        prediction = model.predict(features_scaled)
        return "Default" if prediction[0] == 1 else "No
Default"
    except Exception as e:
        return str(e) # Return the error message for
debugging

# Create Gradio interface with all features
input_features = []
for col in X.columns:
    if X[col].dtype == 'object':
        # For categorical features, use gr.Dropdown
(example, adjust values accordingly)
        unique_values = X[col].unique()

input_features.append(gr.Dropdown(choices=unique_values
, label=col))
    else:
        # For numerical features, use gr.Slider
        min_val = X[col].min()
        max_val = X[col].max()
        input_features.append(gr.Slider(minimum=min_val,
maximum=max_val, value=(min_val + max_val) / 2,
label=col))

# Launch the Gradio interface

```

```

gr.Interface(fn=predict_default, inputs=input_features,
outputs="text").launch().

```

Credit Default Prediction

This model predicts the likelihood of defaulting on a credit obligation. Output meanings:

- Default: Indicates that the person will not be able to pay back their debt in the future.
- No Default: Indicates that the person will be able to pay back their debt in the future.

AGE 30

21 79

PAY_1
1 month late

PAY_2
2 months late

PAY_3
3 months late

PAY_4
3 months late

PAY_5
5 months late

BILL_AMT1 399465.5

-165580 964511

BILL_AMT2 457077

We can predict whether customer able to pay the payment or not.
We can choose the age and can find the accuracy of the payment if
the result is default it indicates the person will not able to pay the
debt in the future.if the results shows not default it it indicates the
person will able to pay the debt in the future.