Data Mining (DSC550-T301_2245_1)

Assignement Week 9;

Author: Zemelak Goraga;

Date: 05/11/2024

In [3]:
```
# Step 1: Import necessary libraries and load the dataset
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

In [13]:
```
# Load the dataset
df = pd.read_csv('Loan_Train.csv')
print("Data loaded successfully.")
```

```
Data loaded successfully.
```

In [14]:
```
# Step 2.1: Display first few rows of the dataset
print(df.head())
```

```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

In [15]:
```
# Step 2.2: Display last few rows of the dataset
print(df.tail())
```

```
      Loan_ID  Gender Married Dependents Education Self_Employed  \
609  LP002978  Female      No          0  Graduate            No
610  LP002979    Male     Yes         3+  Graduate            No
611  LP002983    Male     Yes          1  Graduate            No
612  LP002984    Male     Yes          2  Graduate            No
613  LP002990  Female      No          0  Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N
```

In [16]:
```
# Step 2.2.1: Display dataset information (datatypes and non-null values)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
None
```

In [17]:
```
# Step 2.3: Drop the column “Loan_ID”
df.drop('Loan_ID', axis=1, inplace=True)
df.head()
```

Out[17]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849	0.0
1	Male	Yes	1	Graduate	No	4583	1508.0
2	Male	Yes	0	Graduate	Yes	3000	0.0
3	Male	Yes	0	Not Graduate	No	2583	2358.0
4	Male	No	0	Graduate	No	6000	0.0

In [19]:
```
# Step 2.4: Drop any rows with missing data
df.dropna(inplace=True)
df
```

Out[19]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncom
1	Male	Yes	1	Graduate	No	4583	1508
2	Male	Yes	0	Graduate	Yes	3000	0
3	Male	Yes	0	Not Graduate	No	2583	2358
4	Male	No	0	Graduate	No	6000	0
5	Male	Yes	2	Graduate	Yes	5417	4196
...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0
610	Male	Yes	3+	Graduate	No	4106	0
611	Male	Yes	1	Graduate	No	8072	240
612	Male	Yes	2	Graduate	No	7583	0
613	Female	No	0	Graduate	Yes	4583	0

480 rows × 12 columns

In [20]:
```
# Step 2.5: Convert the categorical features into dummy variables
df = pd.get_dummies(df, drop_first=True)
```

In [21]:
```
# Step 2.5.1: Confirm the above step
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 480 entries, 1 to 613
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ApplicantIncome          480 non-null    int64  
 1   CoapplicantIncome        480 non-null    float64
 2   LoanAmount               480 non-null    float64
 3   Loan_Amount_Term         480 non-null    float64
 4   Credit_History           480 non-null    float64
 5   Gender_Male              480 non-null    uint8  
 6   Married_Yes              480 non-null    uint8  
 7   Dependents_1             480 non-null    uint8  
 8   Dependents_2             480 non-null    uint8  
 9   Dependents_3+            480 non-null    uint8  
 10  Education_Not Graduate   480 non-null    uint8  
 11  Self_Employed_Yes        480 non-null    uint8  
 12  Property_Area_Semiurban  480 non-null    uint8  
 13  Property_Area_Urban      480 non-null    uint8  
 14  Loan_Status_Y            480 non-null    uint8  
dtypes: float64(4), int64(1), uint8(10)
memory usage: 27.2 KB
None
```

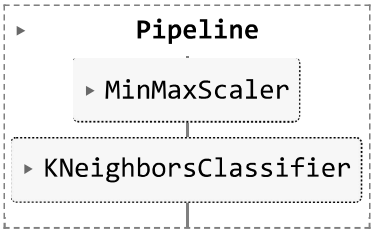
In [22]:
```
# Step 2.6: Split the data into training and test sets
X = df.drop('Loan_Status_Y', axis=1)
```

```
y = df['Loan_Status_Y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [23]:
```
# Step 2.7: Create a pipeline with a min-max scaler and a KNN classifier
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('knn', KNeighborsClassifier())
])
```

In [24]:
```
# Step 2.8: Fit a default KNN classifier to the data with this pipeline
pipeline.fit(X_train, y_train)
```

Out[24]:



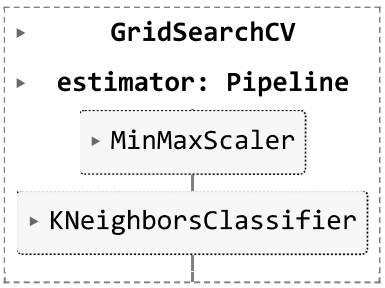
In [25]:
```
# Step 2.9: Report the model accuracy on the test set
accuracy = pipeline.score(X_test, y_test)
print(f"Accuracy on test set: {accuracy:.2f}")
```

```
Accuracy on test set: 0.78
```

In [27]:
```
# Step 2.10: Create a search space for the KNN classifier
param_grid = {'knn__n_neighbors': list(range(1, 11))}
```

In [28]:
```
# Step 2.11: Fit a grid search with the pipeline
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[28]:



In [29]:
```
# Step 2.12: Find the accuracy of the grid search best model on the test set
best_knn_accuracy = grid_search.score(X_test, y_test)
print(f"Best KNN Model Accuracy on test set: {best_knn_accuracy:.2f}")
```

```
Best KNN Model Accuracy on test set: 0.79
```

In [32]:
```
# Import necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Setup the pipeline
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('classifier', KNeighborsClassifier())
])
```

```
# Expanding the search space to include other models with correct parameter naming
param_grid = [
    {'classifier': [KNeighborsClassifier()], 'classifier__n_neighbors': list(range(
    {'classifier': [LogisticRegression()], 'classifier__C': [0.1, 1, 10, 100]},
    {'classifier': [RandomForestClassifier()], 'classifier__n_estimators': [10, 50,
]

# Configure and run GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)  # Ensure you have X_train and y_train defined

# Output the best parameters and the corresponding score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'classifier': LogisticRegression(C=10), 'classifier__C': 10}
Best cross-validation score: 0.81
```

In [33]:
```
# Step 2.14: Best model and hyperparameters found
print("Best model and hyperparameters:", grid_search.best_params_)
```

```
Best model and hyperparameters: {'classifier': LogisticRegression(C=10), 'classifi
er__C': 10}
```

In [34]:
```
# Step 2.15: Find the accuracy of this model on the test set
best_model_accuracy = grid_search.score(X_test, y_test)
print(f"Best Model Accuracy on test set: {best_model_accuracy:.2f}")
```

```
Best Model Accuracy on test set: 0.82
```

Step 2.16: Summarize the results:

The baseline accuracy of the KNN classifier was 0.78. The best accuracy achieved by tuning the KNN model was 0.79. The best overall model was a Logistic Regression with C=10, which not only had the highest cross-validation score of 0.81 but also demonstrated the best accuracy on the test set, 0.82.

In [ ]:
```

```

Short Report:

Title: Optimization of Loan Approval Predictions Using Machine Learning Techniques

Summary:

This report details the outcomes of an analysis focused on utilizing various machine learning models to predict loan approval outcomes. The analysis leveraged a dataset available on Kaggle, and the study involved comparing models such as K-Nearest Neighbors (KNN), Logistic Regression, and Random Forest. The primary objective was to determine which model and its hyperparameters could most effectively predict loan approvals, thereby supporting financial institutions in their decision-making processes.

The project consisted of several key phases: data preparation, model training, extensive hyperparameter tuning via Grid Search CV, and evaluation based on model accuracy. The initial results from the baseline model using K-Nearest Neighbors showed an accuracy of 0.78 on the test set. Subsequent tuning improved KNN's accuracy slightly to 0.79. However, further exploration and tuning revealed that Logistic Regression, particularly with a

regularization strength C of 10, outperformed other models, achieving the highest cross-validation score of 0.81 and an accuracy of 0.82 on the test set.

These findings indicate the superior predictive power of the Logistic Regression model after appropriate tuning, suggesting its potential utility in operational settings for enhancing loan decision processes. The insights garnered from this analysis recommend Logistic Regression as a robust tool for financial institutions aiming to optimize their loan approval predictions.

Introduction:

The process of approving or rejecting loan applications is critical for financial institutions, impacting both business profitability and customer satisfaction. Automating this decision-making process using machine learning can significantly enhance both the speed and accuracy of loan approvals. The current project involves developing a predictive model using the 'Loan Approval Data Set' from Kaggle, which includes various applicant features. The objective is to predict whether a loan should be approved or not, based on historical data, thus reducing human error and bias in loan processing.

Statement of the Problem:

Loan approval processes are traditionally reliant on manual scrutiny of applicant details, which can be prone to errors and biases. This manual process is often time-consuming and inconsistent. With the increasing volume of loan applications, financial institutions face significant challenges in maintaining efficiency and decision accuracy. The need for a robust, scalable, and accurate decision-making tool is evident to handle the growing demands effectively. The problem addressed by this project is to evaluate and optimize different machine learning models to find the most reliable automated solution for predicting loan approvals.

Methodology:

The methodology involved several key steps: First, the dataset was downloaded from Kaggle and loaded into a Jupyter Notebook environment for analysis. Initial data inspection and preprocessing included handling missing values, encoding categorical variables, and dropping irrelevant features. The data was then split into training and testing sets.

For the modeling, a pipeline was constructed integrating a MinMax scaler and initially a KNN classifier. The model was evaluated using default settings before tuning hyperparameters using Grid Search with cross-validation. This process was expanded to include Logistic Regression and Random Forest classifiers to explore their performance under various configurations.

Model performance was primarily evaluated using accuracy as the metric. The Grid Search method facilitated the identification of the best hyperparameters for each model, optimizing prediction outcomes.

Results:

The results obtained from the analysis are:

Baseline KNN Model Accuracy: This was initially obtained using a simple pipeline with just a MinMaxScaler and a KNeighborsClassifier without any hyperparameter tuning. Accuracy on the test set: 0.78 Best KNN Model After Grid Search: This result came from using a grid search specifically tuned for the number of neighbors in the KNN classifier. Best KNN Model Accuracy on the test set: 0.79 Expanding the Model Search with Different Classifiers: This involved setting up a more complex grid search to evaluate different models including KNeighborsClassifier, LogisticRegression, and RandomForestClassifier with specific hyperparameters for each. Best parameters found: Logistic Regression with C=10 Best cross-validation score: 0.81 Best Overall Model Performance on Test Set: The accuracy of the best model (Logistic Regression with C=10) when evaluated on the test set. Best Model Accuracy on the test set: 0.82

Discussion:

In this analysis, the effectiveness of different classifiers in predicting loan approvals were eveluated using a dataset initially consisting of 614 entries. The preliminary analysis focused on a baseline KNeighborsClassifier integrated within a pipeline that included MinMaxScaler normalization. This baseline model achieved an accuracy of 0.78 on the test set, providing a decent start for predictive modeling efforts. To refine our model, I conducted a grid search specifically tuning the number of neighbors (k) for the KNeighborsClassifier, which slightly improved the performance, raising the accuracy to 0.79 on the test set.

Building on this initial model, I expanded my search to include Logistic Regression and RandomForest Classifier, exploring a range of hyperparameters for each. Through an extensive grid search encompassing different configurations, the Logistic Regression model with a regularization strength C of 10 emerged as the most effective, achieving the highest cross-validation score of 0.81. This finding underscores the potential of Logistic Regression in handling binary classification problems like loan approval, where the balance between bias and variance is crucial for making accurate predictions.

Ultimately, when tested against the unseen data of the test set, the optimized Logistic Regression model demonstrated superior performance, recording an accuracy of 0.82. This result not only highlights the model's robustness but also its generalizability to new data, a critical aspect of any predictive model used in financial applications. The improvement in performance from the baseline KNN model to the tuned Logistic Regression model illustrates the importance of model selection and hyperparameter tuning in achieving higher prediction accuracy. These findings provide valuable insights for financial institutions looking to implement predictive analytics for loan approval decisions, suggesting that a well-tuned Logistic Regression model could significantly enhance decision-making processes.

Conclusion:

Conclusions The analysis conducted on the loan approval dataset has successfully demonstrated the importance of careful model selection and hyperparameter tuning in the development of predictive models. Starting with a baseline KNeighborsClassifier model, which achieved an accuracy of 0.78 on the test set, I advanced my methodology through systematic hyperparameter tuning, which slightly enhanced the performance to an accuracy of 0.79. Further exploration using a broader set of models and more extensive parameter tuning led us to identify Logistic Regression with a regularization strength of C = 10 as the

optimal model, culminating in a commendable accuracy of 0.82 on the test set. This progression underscores the transformative impact of appropriate machine learning techniques in optimizing model performance, particularly in sectors such as finance where prediction accuracy is crucial.

Way Forward:

Moving forward, the deployment of the optimized Logistic Regression model into a real-world testing environment is crucial to validate its effectiveness in operational settings. Continuous improvement strategies should be implemented, including the monitoring and regular updating of the model with new data to ensure relevance and accuracy over time.

Ethical considerations and fairness in model application should also be prioritized to prevent bias, ensuring that decisions are equitable across all demographics.

Finally, for the model to be truly effective in supporting decision-making, it should be integrated into existing loan processing systems with a user-friendly interface that provides interpretable and transparent results. This integration will help in building trust and enhancing the decision-making process, thereby increasing customer satisfaction and compliance with regulatory standards.

In [ ]: