# 9.2 Exercise. DSC550 - Jennifer Barrera Conde

May 9, 2024

# 1 9.2 Exercise

## 1.1 DSC550-T301 Data Mining

## 1.2 Jennifer Barrera Conde

### 1.2.1 1. Import the dataset and ensure that it loaded properly.

[1]:
```python
import pandas as pd

# Specify the file path
file_path = 'Loan_Train.csv'

# Check the file path
print("File Path:", file_path)

# I was having issues loading the file for some unknown reason. I ran this to
↪try and find the error. I was able to fix the issue (delete the loaded CSV
↪and upload it to Jupyter again)
try:
    df = pd.read_csv(file_path)
    print("File loaded successfully.")
except pd.errors.EmptyDataError:
    print("The CSV file is empty or does not contain valid data.")
except FileNotFoundError:
    print(f"File not found at path: {file_path}")
except Exception as e:
    print(f"Error occurred while reading the file: {e}")
```

```
File Path: Loan_Train.csv
File loaded successfully.
```

### 1.2.2 2. Prepare the data for modeling by performing the following steps:

```
1. Drop the column "Load_ID."
2. Drop any rows with missing data.
3. Convert the categorical features into dummy variables.
```

[2]:
```python
# Step 1: Drop the "Loan_ID" column
df.drop('Loan_ID', axis=1, inplace=True)

# Step 2: Drop rows with missing data
df.dropna(inplace=True)

# Step 3: Convert categorical features into dummy variables
# Identify:
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
# Convert categorical columns into dummy variables
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Display the first few rows of the processed DataFrame
print(df.head())

# Shape of the DataFrame after processing
print("Shape of DataFrame:", df.shape)
```

```
   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
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
5             5417             4196.0       267.0             360.0

   Credit_History  Gender_Male  Married_Yes  Dependents_1  Dependents_2  \
1             1.0         True         True          True         False
2             1.0         True         True         False         False
3             1.0         True         True         False         False
4             1.0         True        False         False         False
5             1.0         True         True         False          True

   Dependents_3+  Education_Not Graduate  Self_Employed_Yes  \
1          False                   False              False
2          False                   False               True
3          False                    True              False
4          False                   False              False
5          False                   False               True

   Property_Area_Semiurban  Property_Area_Urban  Loan_Status_Y
1                    False                False          False
2                    False                 True           True
3                    False                 True           True
4                    False                 True           True
5                    False                 True           True
Shape of DataFrame: (480, 15)
```

### 1.2.3 3. Split the data into a training and test set, where the "Loan_Status" column is the target.

[3]:
```
from sklearn.model_selection import train_test_split

# Step 1: Prepare the data
# Drop any rows with missing data (NaN)
df.dropna(inplace=True)

# Convert categorical features into dummy variables
df = pd.get_dummies(df, drop_first=True)

# Verify available column names
print("Available columns:", df.columns)

# Step 2: Split the data into features (X) and target (y)
X = df.drop('Loan_Status_Y', axis=1)  # Features (all columns except the target)
y = df['Loan_Status_Y']               # Target variable ('Loan_Status_Y')

# Step 3: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Display the shapes of the resulting datasets
print("Results of the Training and Testing:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
Available columns: Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Gender_Male', 'Married_Yes',
       'Dependents_1', 'Dependents_2', 'Dependents_3+',
       'Education_Not Graduate', 'Self_Employed_Yes',
       'Property_Area_Semiurban', 'Property_Area_Urban', 'Loan_Status_Y'],
      dtype='object')
Results of the Training and Testing:
X_train shape: (384, 14)
X_test shape: (96, 14)
y_train shape: (384,)
y_test shape: (96,)
```

### 1.2.4 4. Create a pipeline with a min-max scaler and a KNN classifier (see section 15.3 in the Machine Learning with Python Cookbook).

[4]:
```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

# Create a pipeline
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),          # MinMaxScaler for feature scaling
    ('classifier', KNeighborsClassifier())  # K-Nearest Neighbors classifier
])

# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predict on the test data
y_pred = pipeline.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("The result is:")
print(f"Accuracy: {accuracy:.2f}")
```

```
The result is:
Accuracy: 0.78
```

### 1.2.5 5. Fit a default KNN classifier to the data with this pipeline. Report the model accuracy on the test set. Note: Fitting a pipeline model works just like fitting a regular model.

[5]:
```python
# Evaluate the model accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
```

```
Model Accuracy on Test Set: 0.7812
```

### 1.2.6 6. Create a search space for your KNN classifier where your "n_neighbors" parameter varies from 1 to 10. (see section 15.3 in the Machine Learning with Python Cookbook).

[11]:
```python
from sklearn.model_selection import train_test_split, GridSearchCV

# Define the parameter grid for GridSearchCV
param_grid = {
    'classifier__n_neighbors': list(range(1, 11))  # Search space for␣
 ↪n_neighbors from 1 to 10
```

```
}

# Create GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit GridSearchCV on the training data
grid_search.fit(X_train, y_train)

# Get the best model and evaluate on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the best model's accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print(f"Best Model Accuracy on Test Set: {accuracy:.4f}")
print(f"Best Parameters: {grid_search.best_params_}")
```

```
Best Model Accuracy on Test Set: 0.7917
Best Parameters: {'classifier__n_neighbors': 3}
```

### 1.2.7 7. Fit a grid search with your pipeline, search space, and 5-fold cross-validation to find the best value for the "n_neighbors" parameter.

[12]:
```
# Get the best model and its parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
```

```
Best Parameters: {'classifier__n_neighbors': 3}
Model Accuracy on Test Set: 0.7917
```

### 1.2.8 8. Find the accuracy of the grid search best model on the test set. Note: It is possible that this will not be an improvement over the default model, but likely it will be.

[13]:
```
# Fit GridSearchCV on the training data
grid_search.fit(X_train, y_train)

# Get the best model and its parameters
best_model = grid_search.best_estimator_

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
```

```
Model Accuracy on Test Set: 0.7917
```

### 1.2.9 9. Now, repeat steps 6 and 7 with the same pipeline, but expand your search space to include logistic regression and random forest models with the hyperparameter values in section 12.3 of the Machine Learning with Python Cookbook.

[16]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

# Define the parameter grid for GridSearchCV
param_grid = [
    {
        'classifier': [KNeighborsClassifier()],
        'classifier__n_neighbors': range(1, 11)  # Search space for KNN's␣
 ↪n_neighbors from 1 to 10
    },
    {
        'classifier': [LogisticRegression()],
        'classifier__C': [0.1, 1.0, 10.0]          # Regularization parameter␣
 ↪for Logistic Regression
    },
    {
        'classifier': [RandomForestClassifier()],
        'classifier__n_estimators': [100, 200],     # Number of trees in the␣
 ↪forest
        'classifier__max_depth': [None, 5, 10]       # Maximum depth of each␣
 ↪tree in the forest
    }
]

# Create GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit GridSearchCV on the training data
grid_search.fit(X_train, y_train)

# Results:
# Get the best model and its parameters
best_model = grid_search.best_estimator_
```

```
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
```

```
Best Parameters: {'classifier': LogisticRegression(C=10.0), 'classifier__C':
10.0}
Model Accuracy on Test Set: 0.8229
```

### 1.2.10 10. What are the best model and hyperparameters found in the grid search? Find the accuracy of this model on the test set.

### 1.2.11 11. Summarize your results.

The best model identified is LogisticRegression classifier with the hyperparameter C set to 10.0. The accuracy of this best model on the test set is approximately 0.8229 (or 82.29%).

This indicates that among the models tested (K-Nearest Neighbors, Logistic Regression, Random Forest), the LogisticRegression model with a regularization strength (C) of 10.0 achieved the highest accuracy on unseen test data.

The logistic regression model is the most effective choice for this particular dataset and task, based on the evaluation using cross-validated grid search and subsequent testing on the hold-out test data.

[ ]:
```

```