DSC550-T301

Chitramoy Mukherjee

Week-9

Date: 2/6/2024

Exercise - 9.2

Import the Loan Approval Data Set and ensure that it loaded properly

In [58]:

```
import warnings
warnings.filterwarnings('ignore')

# Required python basic libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from math import sqrt
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

#Required python visualization libraries

# import missingno as msno
import matplotlib
import matplotlib.pyplot as plt

def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)

### Reading the mxmh_survey_results.csv dataset
loan_df = pd.read_csv("C://Users//14024//OneDrive//Desktop//MS-DSC//DSC-550/Week-9//Loan_Train.cs

# Check first 5 rows of the dataset
loan_df.head()
```

Out[58]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmour
0	LP001002	Male	No	0	Graduate	No	5849	0.0	Nal
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.

Drop the "Loan_ID" column and drop any rows with missing data and Convert the categorical features into dummy variables.

In [59]:

```
# Drop the "Load_ID" column
loan_df = loan_df.drop("Loan_ID", axis=1)

# Drop rows with missing data
df = loan_df.dropna()

# Display the first few rows of the DataFrame with dummy variables
print(df.head())
```

```
  Gender Married Dependents     Education Self_Employed  ApplicantIncome  \
1   Male     Yes          1      Graduate            No             4583   
2   Male     Yes          0      Graduate           Yes             3000   
3   Male     Yes          0  Not Graduate            No             2583   
4   Male      No          0      Graduate            No             6000   
5   Male     Yes          2      Graduate           Yes             5417   

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0   
2                0.0        66.0             360.0             1.0   
3             2358.0       120.0             360.0             1.0   
4                0.0       141.0             360.0             1.0   
5             4196.0       267.0             360.0             1.0   

  Property_Area Loan_Status  
1         Rural           N  
2         Urban           Y  
3         Urban           Y  
4         Urban           Y  
5         Urban           Y  
```

In [60]:

```
# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

# Create dummy variables for categorical columns
df_dummies = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Display the first few rows of the DataFrame with dummy variables
print(df_dummies.head())
```

```
   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
```

Split the data into a training and test set, where the “Loan_Status” column is the target

In [61]:

```
# Identify features (X) and target variable (y)
X = df.drop(columns=['Loan_Status'])
y = df['Loan_Status']

# 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 shape of the training and test sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (384, 11) (384,)
Test set shape: (96, 11) (96,)
```

#### Create a pipeline with a min-max scaler and a KNN classifier.Fit a default KNN classifier to the data with this pipeline. Report the model accuracy on the test set.

In [62]:

```
# Identify the numerical and categorical columns
numerical_features = X.select_dtypes(include=['number']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Identify categorical columns in the dataset
categorical_columns = X_train.select_dtypes(include=['object']).columns

# Create a column transformer to apply different preprocessing to numerical and categorical colum
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_columns)
    ])

# Create a pipeline with a Min-Max Scaler, One-Hot Encoder, and a KNN Classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),    # Step 1: Preprocessing
    ('Classifier', KNeighborsClassifier())     # Step 2: KNN Classifier
])

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

# predict the test set
y_pred = pipeline.predict(X_test)

# Calculate and report the model accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.78125
```

#### Create a search space for your KNN classifier where your “n_neighbors” parameter varies from 1 to 10.

In [63]:

```
# Create a ColumnTransformer to preprocess numerical and categorical features separately
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_columns)
    ])

# Create a pipeline with a Min-Max Scaler, One-Hot Encoder, and a KNN Classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),    # Step 1: Preprocessing
    ('Classifier', KNeighborsClassifier())     # Step 2: KNN Classifier
])

# Create a search space for the KNN classifier
param_grid = {
    'Classifier__n_neighbors': list(range(1, 11))
}

# Create a GridSearchCV object
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and model accuracy
best_n_neighbours = grid_search.best_params_['Classifier__n_neighbors']

# predict on the test set using best model
y_pred = grid_search.predict(X_test)

#Accuracy
accuracy
```

Out[63]:

```
0.78125
```

#### Fit a grid search with your pipeline, search space, and 5-fold cross-validation to find the best value for the “n_neighbors” parameter. Find the accuracy of the grid search best model on the test set.

In [64]:

```python
# Create a ColumnTransformer to preprocess numerical and categorical features separately
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_columns)
    ])

# Create a pipeline with a Min-Max Scaler, One-Hot Encoder, and a KNN Classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),    # Step 1: Preprocessing
    ('Classifier', KNeighborsClassifier())     # Step 2: KNN Classifier
])

# Define the search space for n_neighbours parameter
param_grid = {'Classifier__n_neighbors': [1,3,5,7,9]}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Step 9: Find the best model from the grid search
best_n_neighbours = grid_search.best_params_['Classifier__n_neighbors']

# Step 10: Find the accuracy of the grid search best model on the test set
y_pred = grid_search.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Best n_neighbours parameter: {best_n_neighbours}")
print(f"Accuracy with Best Estimator:", accuracy)
```

```
Best n_neighbours parameter: 9
Accuracy with Best Estimator: 0.75
```

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 [66]:

```python
# Create a pipeline with the preprocessor and a classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', None)  # Placeholder for the classifier
])

# Define the search space for classifiers and their hyperparameters
param_grid = [
    {
        'classifier': [KNeighborsClassifier()],
        'classifier__n_neighbors': range(1, 11)
    },
    {
        'classifier': [LogisticRegression()],
        'classifier__C': [0.1, 1, 10],
        'classifier__max_iter': [100, 200, 300]
    },
    {
        'classifier': [RandomForestClassifier()],
        'classifier__n_estimators': [50, 100, 150],
        'classifier__max_depth': [None, 10, 20],
        'classifier__min_samples_split': [2, 5, 10]
    }
]

# Create the GridSearchCV object
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

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

# Get the best classifier and hyperparameters
best_classifier = grid_search.best_params_['classifier']
best_hyperparameters = {key.replace('classifier__', ''): value for key, value in grid_search.best_

# Predict on the test set using the best model
y_pred = grid_search.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Best Classifier: {best_classifier}")
print(f"Best Hyperparameters: {best_hyperparameters}")
print("Model Accuracy on Test Set:", accuracy)
```

```
Best Classifier: LogisticRegression(C=10)
Best Hyperparameters: {'C': 10, 'max_iter': 100}
Model Accuracy on Test Set: 0.8229166666666666
```

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

In [67]:

```
# Get the best classifier and hyperparameters
best_classifier = grid_search.best_params_['classifier']
best_hyperparameters = {key.replace('classifier__', ''): value for key, value in grid_search.best_

# Predict on the test set using the best model
y_pred = grid_search.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Best Classifier: {best_classifier}")
print(f"Best Hyperparameters: {best_hyperparameters}")
print("Model Accuracy on Test Set:", accuracy)

# Overall result is not much different but result compared to default model has improved.
```

```
Best Classifier: LogisticRegression(C=10)
Best Hyperparameters: {'C': 10, 'max_iter': 100}
Model Accuracy on Test Set: 0.8229166666666666
```

#### Summarize your results.

1.   The initial steps involved data preparation, including handling missing values and encoding categorical variables.

2.   The default KNN classifier was used to establish a baseline accuracy on the test set.

3.   The grid search for the KNN classifier helped identify the best value for the "n_neighbors" parameter, potentially improving the model's performance. In our solution, we have we used GridSearchCV to conduct five-fold cross-validation on KNN classifier with different value of K. When that is completed , we can see the K that produces the best model have value 9.

4. Model Accuracy with Best Parameters is 78.12% and Accuracy of the Best Model on Test Set: 75.00%.This accuracy represents the performance of the model on the validation set after the hyperparameter tuning.The validation set is typically a subset of the training data that is reserved for fine-tuning the model.An accuracy of 78.12% suggests that the model, with the tuned hyperparameters, correctly predicted the target variable for approximately 78.12% of the instances in the validation set.An accuracy of 75.00% indicates that the model performed well on new, unseen data, correctly predicting the target variable for approximately 75% of the instances in the test set.The fact that the accuracy on the test set is slightly higher than the accuracy with the best parameters on the validation set is a positive sign, suggesting that the model generalizes well to new data.The overall result suggest that the hyperparameter tuning process, which involved optimizing the "n_neighbors" parameter for the KNN model, contributed to an improvement in the model's predictive performance. The model appears to generalize well to new, unseen data based on the higher accuracy observed on the test set compared to the validation set accuracy with the best parameters.

5.   After employing the grid search across Logistic regression, KNN and random forest, Logistic regressiopn looks like the optimal classifier.Identified hyperparameters include a C value of 10 and max iteration of 100. Model accuarcy increased to 82% on test dataset.

6.   The best model and hyperparameters were identified based on the grid search results, and the accuracy of this model on the test set was reported.