C550-T301-Data_mining_2241_week9_Samanta_rajib

October 30, 2023

0.1 Class : C550-T301 Data Mining (2241-1)

0.2 Name : Rajib Samanta

0.2.1 Assignment : Week 9

In this exercise, you will work with the Loan_Train.csv dataset which can be downloaded from this link: Loan Approval Data Set.

1. Import the dataset and ensure that it loaded properly.
2. Prepare the data for modeling by performing the following steps:
   a. Drop the column "Load_ID."
   b. Drop any rows with missing data.
   c. Convert the categorical features into dummy variables.
3. Split the data into a training and test set, where the "Loan_Status" column is the target.
4. Create a pipeline with a min-max scaler and a KNN classifier (see section 15.3 in the Machine Learning with Python Cookbook).
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.
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).
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.
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.
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.
10. What are the best model and hyperparameters found in the grid search? Find the accuracy of this model on the test set.
11. Summarize your results.

[142]:
```
# Import Libraries
import pandas as pd
import os
#pip install textblob
#from textblob import TextBlob
# pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import re
```

```python
from sklearn.model_selection import train_test_split, GridSearchCV
import numpy as np
#from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score, mean_squared_error
#from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

[143]:

```python
# Read the Loan Approval Data Set file ('Loan_Train.csv') from local:
directory = '/Users/rajibsamanta/Documents/Rajib/College/Sem6_fall_2023/Week9'
# Set the working directory
os.chdir(directory)
print(os.getcwd())
# 1. Import the movie review data as a data frame and ensure that the data is␣
↪loaded properly.

file_name = "Loan_Train.csv"

# Load the dataset into a pandas DataFrame
df = pd.read_csv(file_name)

# Display few records.
df.head()
```

```
/Users/rajibsamanta/Documents/Rajib/College/Sem6_fall_2023/Week9
```

[143]:

```
    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
```

[144]:
```python
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
```

[145]:
```python
# Drop the 'Load_ID' column
# create a new DataFrame without modifying the original one, use:
new_df = df.drop('Loan_ID', axis=1)
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             601 non-null    object
 1   Married            611 non-null    object
 2   Dependents         599 non-null    object
 3   Education          614 non-null    object
 4   Self_Employed      582 non-null    object
 5   ApplicantIncome    614 non-null    int64
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object
```

```
 11  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

[146]:
```
# 2. Drop rows with missing data
new_df.dropna(inplace=True)
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 480 entries, 1 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             480 non-null    object
 1   Married            480 non-null    object
 2   Dependents         480 non-null    object
 3   Education          480 non-null    object
 4   Self_Employed      480 non-null    object
 5   ApplicantIncome    480 non-null    int64
 6   CoapplicantIncome  480 non-null    float64
 7   LoanAmount         480 non-null    float64
 8   Loan_Amount_Term   480 non-null    float64
 9   Credit_History     480 non-null    float64
 10  Property_Area      480 non-null    object
 11  Loan_Status        480 non-null    object
dtypes: float64(4), int64(1), object(7)
memory usage: 48.8+ KB
```

[147]:
```
# Convert the categorical features into dummy variables.
# Identify and list categorical features
categorical_columns = new_df.select_dtypes(include=['object', 'category']).
 ↪columns.tolist()

# The 'categorical_features' list will now contain the names of the categorical␣
 ↪columns

print(categorical_columns)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
'Property_Area', 'Loan_Status']
```

[148]:
```
# Convert categorical columns to dummy variables
new_df = pd.get_dummies(new_df, columns=categorical_columns, drop_first=True)
new_df.head()
```

[148]:
```
   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            1            1             1             0
2             1.0            1            1             0             0
3             1.0            1            1             0             0
4             1.0            1            0             0             0
5             1.0            1            1             0             1

   Dependents_3+  Education_Not Graduate  Self_Employed_Yes  \
1              0                       0                  0
2              0                       0                  1
3              0                       1                  0
4              0                       0                  0
5              0                       0                  1

   Property_Area_Semiurban  Property_Area_Urban  Loan_Status_Y
1                        0                    0              0
2                        0                    1              1
3                        0                    1              1
4                        0                    1              1
5                        0                    1              1
```

[149]:
```
new_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
```

Orginal dataset had 614 rows after ckeaning its now 480

[150]:
```
# 3. Split the data into a training and test set, where the "Loan_Status"␣
↪column is the target.
# Split the data into features (X) and the target variable (y)
X = new_df.drop(columns=['Loan_Status_Y'])
y = new_df['Loan_Status_Y']

# Split the data into a training set and a test set (e.g., 75% training, 25%␣
↪testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)
```

[151]:
```
# 4. Create a pipeline with a min-max scaler and a KNN classifier

pipeline = Pipeline([
    ('scaler', MinMaxScaler()),  # Min-Max Scaler
    ('classifier', KNeighborsClassifier(n_neighbors=5))  # KNN Classifier␣
↪(desired number of neighbors is 5 here)
])

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

# Make predictions on the test data
y_pred = pipeline.predict(X_test)
```

[152]:
```
y_pred
```

[152]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=uint8)
```

[153]:
```
# 5. Fit a default KNN classifier to the data with this pipeline. Report the␣
↪model accuracy on the test set.
# Calculate the model accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)

# Report the model accuracy
print(f'Model Accuracy on Test Set: {accuracy:.2f}')
```

```
Model Accuracy on Test Set: 0.73
```

[154]:
```
#6. Create a search space for your KNN classifier where your "n_neighbors"␣
↪parameter varies from 1 to 10.
# Create a KNN classifier
knn_classifier = KNeighborsClassifier()
# Define the hyperparameter search space for 'n_neighbors'
param_grid = {
    'classifier__n_neighbors': list(range(1, 11))  # Vary 'n_neighbors' from 1␣
↪to 10
}
```

[155]:
```
param_grid
```

[155]: {'classifier__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}

[156]:
```
# 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

# Create a GridSearchCV object with 5-fold cross-validation
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,␣
↪scoring='accuracy', cv=5)

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

# Get the best 'n_neighbors' value from the search
best_n_neighbors = grid_search.best_params_['classifier__n_neighbors']

# Report the best 'n_neighbors' value
print(f'Best n_neighbors: {best_n_neighbors}')
```

Best n_neighbors: 3

[157]:
```
# 8. Find the accuracy of the grid search best model on the test set.
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_model.predict(X_test)

# Calculate the accuracy of the best model on the test set
accuracy = accuracy_score(y_test, y_pred)

# Report the accuracy of the best model
print(f'Accuracy of the Best Model on Test Set: {accuracy:.2f}')
```

Accuracy of the Best Model on Test Set: 0.75

[158]:
```
# 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 value
# Split the data into features (X) and the target variable (y)
#X = df_new.drop(columns=['Loan_Status'])
#y = df_new['Loan_Status']
# Split the data into a training set and a test set (e.g., 80% training, 20%␣
↪testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

## Create a pipeline with a Standard Scaler, Min-Max Scaler, and classifier
pipeline = Pipeline([
    ('std_scaler', StandardScaler()),  # Standard Scaler
    ('min_max_scaler', MinMaxScaler()),  # Min-Max Scaler
    ('classifier', KNeighborsClassifier())  # KNN Classifier (default, will be␣
↪replaced during GridSearch)
])

# Define the hyperparameter search space for classifiers
param_grid = {
    'classifier': [KNeighborsClassifier(), LogisticRegression(),␣
↪RandomForestClassifier()],
    'classifier__n_neighbors': list(range(1, 11)),  # KNN parameter search
    'classifier__C': [0.01, 0.1, 1.0, 10.0],  # Logistic Regression parameter␣
↪search
    'classifier__n_estimators': [50, 100, 200],  # Random Forest parameter␣
↪search
}

# Create a GridSearchCV object with 5-fold cross-validation
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,␣
↪scoring='accuracy', cv=5)
```

[159]:
```
param_grid
```

[159]:
```
{'classifier': [KNeighborsClassifier(),
  LogisticRegression(),
  RandomForestClassifier()],
 'classifier__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 'classifier__C': [0.01, 0.1, 1.0, 10.0],
 'classifier__n_estimators': [50, 100, 200]}
```

[160]:
```
##--> Random Forest parameter search
# Split the data into a training set and a test set (e.g., 80% training, 20%␣
↪testing)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create a pipeline with a Standard Scaler, Min-Max Scaler, and classifier
pipeline = Pipeline([
    ('std_scaler', StandardScaler()),  # Standard Scaler
    ('min_max_scaler', MinMaxScaler()),  # Min-Max Scaler
    ('classifier', KNeighborsClassifier(n_neighbors=3))  # KNN Classifier␣
↪(default, will be replaced during GridSearch)
])

# Define the hyperparameter search space for classifiers
param_grid = {
    'classifier':[RandomForestClassifier()],
    'classifier__n_estimators': [50, 100, 200],  # Random Forest parameter␣
↪search
}

# Create a GridSearchCV object with 5-fold cross-validation
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,␣
↪scoring='accuracy', cv=5)

# Fit the GridSearchCV object on the training data
grid_search.fit(X_train, y_train)
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_model.predict(X_test)

# Calculate the accuracy of the best model on the test set
accuracy = accuracy_score(y_test, y_pred)

# Report the accuracy of the best model
print(f'Accuracy of the Best Model on Test Set: {accuracy:.2f}')
# Get the best 'n_neighbors' value from the search
best_n_neighbors = grid_search.best_params_['classifier__n_estimators']

# Report the best 'n_neighbors' value
print(f'Best n_neighbors: {best_n_neighbors}')
```

```
Accuracy of the Best Model on Test Set: 0.81
Best n_neighbors: 100
```

[161]:
```
## --> Logistic Regression parameter search
# Split the data into a training set and a test set (e.g., 80% training, 20%␣
↪testing)
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a pipeline with a Standard Scaler, Min-Max Scaler, and classifier
pipeline = Pipeline([
    ('std_scaler', StandardScaler()),  # Standard Scaler
    ('min_max_scaler', MinMaxScaler()),  # Min-Max Scaler
    ('classifier', KNeighborsClassifier(n_neighbors=5))  # KNN Classifier (default, will be replaced during GridSearch)
])

# Define the hyperparameter search space for classifiers
param_grid = {
    'classifier':[LogisticRegression(solver='liblinear')],
    'classifier__C': [0.01, 0.1, 1.0, 10.0],  # Logistic Regression parameter search
}

# Create a GridSearchCV object with 5-fold cross-validation
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the GridSearchCV object on the training data
grid_search.fit(X_train, y_train)
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_model.predict(X_test)

# Calculate the accuracy of the best model on the test set
accuracy = accuracy_score(y_test, y_pred)

# Report the accuracy of the best model
print(f'Accuracy of the Best Model on Test Set: {accuracy:.2f}')
# Get the best 'n_neighbors' value from the search
best_n_neighbors = grid_search.best_params_['classifier__C']

# Report the best 'n_neighbors' value
print(f'Best n_neighbors: {best_n_neighbors}')
```

```
Accuracy of the Best Model on Test Set: 0.82
Best n_neighbors: 10.0
```

Here is the summary of the result:

a. KNN classifier , Best n_neighbors: 3 & Accuracy on Test Set: 0.75
b. Random Forest parameter search ,Best n_neighbors: 100 & Accuracy on Test Set: 0.81

c. Logistic Regression ,Best n_neighbors: 10.0 & Accuracy on Test Set: 0.82

Logistic Regression & Random Forest parameter search if better than KNN classifier grid search. Accuracy around 0.82 whereas KNN normal grid search accurecy is 0.75

```
[ ]:
```