Credit Risk Analysis (P2P lending platform)

Pipeline Creation Report

platform: Bondora

by: Nour M. Ibrahim

In this Report, I'll illustrate the process and python modules we used to build the pipeline object for our task.

Which are:

1. Split the data into training and testing.
2. Classification Pipeline Creation.
3. Regression Pipeline Creation.

A Pipeline is a procedure used to execute a workflow of machine learning modules and techniques in a pipe-like manner.

In this project, we're using Python to execute our work, from which Scikit-learn library has all the necessary features and modules to handle the creation of a Pipeline and Its components.

I. Split the data into training and testing:

1) Split into training and testing using train_test_split module from sklearn.model_selection.
   Test data size = 25 % of the data
   random_state = 0

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

2) Split target variable y in both train and test data into classification, and regression target variables.
   Classification target → LoanStatus → position 0 in y_train and y_test.
   Regression targets → [EMI, ELA, PROI] → positions 1, 2, and 3 in y_train, and y_test.

```
# Separating Target values for classifications and regression problems
y_class_train = y_train.iloc[:,0]
y_reg_train = y_train.iloc[:,1:]

y_class_test = y_test.iloc[:,0]
y_reg_test = y_test.iloc[:,1:]
```

II. Classification Pipeline:

1) Using y_class_train, and y_class_test (classification target variable)
2) Data Preprocessing by Standard Scaler (normal distribution based)
3) Reduce Data Dimension using PCA model. (n_components = 110)
4) Apply Classifier,
   - Using Logistic Regression Classification Model from sklearn.linear_model.
5) Create the Pipeline object.

```
# Create Pipeline
pipeline_class = Pipeline([
    ('stdscaler', StandardScaler()),
    ('pca', PCA(n_components=110)),
    ('Classifier', LogisticRegression(random_state=0))
])
```

6) Fit the Pipeline object on the training data, and predict the target variable based on test data.

```
# fit and transform the pipeline
pipeline_class.fit(X_train, y_class_train)

# predict using the pipeline
pred_class = pipeline_class.predict(X_test)
```

7) Apply accuracy score, confusion metric, and classification report metrics from sklearn.metrics to assess the performance of the Pipeline.

```
print("Logistic Regression:")

print("\nAccuracy score:\n", round(accuracy_score(y_class_test, pred_class), 3))
print('*'*40)
print("\nConfusion Matrix:\n", confusion_matrix(y_class_test, pred_class))
print('*'*40)
print("\nClassification Report:\n", classification_report(y_class_test, pred_class,
                                    target_names=['Default', 'Non-deafault']))
```

The results of the metrics indicate that workflow is a very good fit for the task in hand,

```
Logistic Regression:


Accuracy score:
 89.33 %
****************************************
```

Since the accuracy score isn't the perfect metric in a binary target classification problem, we used a confusion matrix, and classification report metrics also,

```
Confusion Matrix:
 [[9023 1656]
 [ 408 8262]]
**************************************

Classification Report:
              precision    recall  f1-score   support

     Default       0.96      0.84      0.90     10679
Non-deafault       0.83      0.95      0.89      8670

    accuracy                           0.89     19349
   macro avg       0.89      0.90      0.89     19349
weighted avg       0.90      0.89      0.89     19349
```

The recall of No-default loans is 0.95, while it's 0.84 for default loans.

III. Regression Pipeline:

1) Using y_reg_train and y_reg_test
2) Data Preprocessing using the Standard Scaler module from sklearn.preprocessing
3) Reduce Data Dimension using the PCA module from sklearn.decomposition
4) Apply Regression Model,
   - Using Ridge Regression Model from sklearn.linear_model.
5) Create the Pipeline object.

```
model_reg = Pipeline([
    ('stdscaler', StandardScaler()),
    ('pca', PCA(n_components=110)),
    ('Regressor', Ridge(random_state=0))
])
```

6) Fit the Pipeline object on the training data and predict the target variable based on test data.

```
# fit and transform the pipeline
model_reg.fit(X_train, y_reg_train)

# predict using the pipeline
pred_reg = model_reg.predict(X_test)
```

7) Apply R2 Score metric from sklearn.metrics to access the performance of the model.

```
# Score and test results
print('R2_score : ', round(r2_score(y_reg_test, pred_reg)*100,2), '%')
```

```
R2_score :  91.65 %
```

IV. Saving the Pipelines:

Using pickle module, we save the model with dump() function

```
pickle.dump(pipeline_class, open('pipeline_class.pkl', 'wb'))
pickle.dump(pipeline_reg, open('pipeline_reg.pkl', 'wb'))
```

✓ 0.3s

This concludes the Report.

Best Regards…