Assignment 5

FE 520 - Into to Python for Financial Applications

Parth Parab
MS CS '21
10444835
December 16, 2020

Q1. Time Series Data Practice

We start by importing numpy and pandas as two standard packages to solve this problem. Next, we import *Energy.xlsx* as pandas data-frame. We then define a function **splitEnergy()** which takes in two values *StartYear* and *EndYear*. Both have default values *2012* and *None* respectively. We start the function definition by setting the 'Data Date' column of the data-frame to type string for easy manipulation and then splitting just the year and converting it back to integer again. The function definition has one if-else as follows -

- If the EndYear is None then the TestData gets all the values with StartYear and TrainData gets the rest
- By default, all company Data Date within 2012 will be selected as Testing data
- Else TestData gets all values within range of StartYear and EndYear and TrainData gets everything else.

TestData and TrainData are then only given columns from *Accumulated Other Comprehensive Income (Loss)* to *Selling, General and Administrative Expenses* by getting their column index by name and then dropping every other column which is not in that range.

The function then returns TestData.values and TrainData.values which in the form of numpy array.

Results

```
splitEnergy(2012,2013)
```

```
(array([[-1057.,  1421.,  8212., ...,     0.,     0.,    nan],
        [ -779.,  1699.,  7925., ...,     0.,     0.,    nan],
        [ -675.,  2167.,  8157., ...,     0.,     0.,    nan],
        ...,
        [ -254.,   190., 14252., ...,     0.,    nan,    nan],
        [ -205.,   206., 13531., ...,     0.,    nan,    nan],
        [ -204.,   197., 12737., ...,     0.,    nan,    nan]]),
 array([[-1749.,  1502.,  8780., ...,     0.,     0.,    nan],
        [-1603.,  1434.,  8296., ...,     0.,     0.,    nan],
        [-1377.,   865.,  8839., ...,     0.,     0.,    nan],
        ...,
        [ -321.,   142., 10304., ...,    nan,    nan,    nan],
        [ -327.,   176.,  9545., ...,    nan,    nan,    nan],
        [ -234.,   236., 10401., ...,    nan,    nan,    nan]]))
```

Final output of Q1

Q3. Regression

3.1 In this question, we are going to use the diabetes data set. Use sklearn.datasets.load diabetes() to load the data and labels.

We import the diabetes datasets in two variables X and y by passing the argument *return_X_y=True* in **sklearn.datasets.load_diabetes()**

3.2. Randomly split the data into training set (80%) and testing set (20%)

Here we use *train_test_split* from sklearn. We pass two specific arguments in the function -

- test_size = 0.2 which gives 20% to test data and 80% to training data
- random_state = 42 which controls the shuffling(to randomize) applied to the data before applying the split.

3.3. Create a linear regression model using sklearn, and fit training data. Evaluate your model using test data. Give all the coefficient and R-squared score.

We import Linear Regression from sklearn.model. We then pass the training data (diabetes_X_train, diabetes_y_train) for the algorithm to train on (through the fit function) and then perform a prediction (using the predict function) on the testing data (diabetes_X_test). We then use the .coef_ function of the algorithm to fetch all the coefficients and finally compare the prediction with our sample space (diabetes_y_test) to calculate the R-squared score.

Result

```
Coefficients:
 [  37.90031426 -241.96624835  542.42575342  347.70830529 -931.46126093
  518.04405547  163.40353476  275.31003837  736.18909839   48.67112488]

Coefficient of determination: 0.45
```

3.4. Use 10-fold cross validation to fit and validate your linear regression models on the whole data set. Print the scores for each validation.

We run K-Cross Validation (k=10) on out Linear Regression model and print the CV score for each iteration. The final output is the mean of the absolute scores (accuracy) along with the standard deviation.

Result

```
Cross Validation Score for fold  1 is:  0.5310818612097871
Cross Validation Score for fold  2 is:  0.48929200594262445
Cross Validation Score for fold  3 is:  0.6155148865078137
Cross Validation Score for fold  4 is:  -0.07802446434722854
Cross Validation Score for fold  5 is:  0.4650954815453411
Cross Validation Score for fold  6 is:  0.5338119174559128
Cross Validation Score for fold  7 is:  0.706563282930452
Cross Validation Score for fold  8 is:  0.5253790639706339
Cross Validation Score for fold  9 is:  -0.23181569661350765
Cross Validation Score for fold  10 is:  0.37529064060113176

Mean Accuracy: 0.46 with STD DEVIATION:  0.58
```

3.5. Use sklearn to create RandomForestRegressor model, fit the training data into it.

We import RandomForestRegressor from sklearn and pass it the following arguments -

- n_estimators=1000 - The number of trees in the forest.
- max_depth=None - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_split=2 - The minimum number of samples required to split an internal node
- random_state=1 - Controls both the randomness of the bootstrapping of the samples used when building trees

We then calculate and print the *R-squared score* and *Root mean squared test error*

Result

```
R-squared score: 0.44
Root mean squared test error = 54.465358204535065
```

2.6. Use Grid Search to find the optimal hyper-parameters (max depth:{None, 7, 4} and min samples split: {2, 10, 20}) for RandomForestRegressor.

We start by setting up the **RandomForestRegressor** with the values used in the previous question. We then setup **GridSearchCV** with the parameters mentioned in the question and cv=10 and train the model.

The model gives best *max_depth = None* and *min_samples_split = 20*

Result

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 20,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 1000,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 1,
 'verbose': 0,
 'warm_start': False}
```

We further calculate the *R-squared score* using the optimal parameters which comes out to be 0.46 which is better than RandomForestRegressor without GridSearch (0.44)

CODE APPENDIX

Q1

```
#!/usr/bin/env python
# coding: utf-8

# In[223]:


#import statements
import pandas as pd
import numpy as np


# In[224]:


#import Energy.xlsx to pandas dataframe
df = pd.read_excel (r'Energy.xlsx')
print(df.shape) #check if data is imported correctly by checking dimensions


# In[225]:


#getting column index by column name
index_1 = df.columns.get_loc("Accumulated Other Comprehensive Income (Loss)")
index_2 = df.columns.get_loc("Selling, General and Administrative Expenses")


# In[226]:


#function definition for splitEnergy
def splitEnergy(StartYear=2012,EndYear=None):
    df_temp = df

    #set column 'Data Date' type to string to split year and convert back to 
integer
    df_temp['Data Date'] = df_temp['Data Date'].astype(str).str[:4].astype(int)

    #If EndYear is None, we will only choose all data with ”Data Date” == 
StartYear as Test data, all other data as Train data.
```

```
        #By default, all company Data Date within 2012 will be selected as Testing
data.
        if (EndYear == None):
            TestData = df_temp[df_temp['Data Date'] == StartYear]
            TrainData = df_temp[df_temp['Data Date'] != StartYear]


        #If EndYear is NOT None, we will choose all data with "Data Date" == StartYear
to EndYear as Test data, all other data as Train data
        else:
            TestData = df_temp[(df_temp['Data Date'] >= StartYear) & (df_temp['Data
Date'] <= EndYear)]
            TrainData = df_temp[(df_temp['Data Date'] < StartYear) | (df_temp['Data
Date'] > EndYear)]

        #array from column "Accumulated Other Comprehensive Income (Loss)" to column "
Selling, General and Administrative Expenses".
        TestData = TestData.iloc[:, index_1:index_2]
        TrainData = TrainData.iloc[:, index_1:index_2]

        #Output Data type: Array(Numpy)
        return TestData.values , TrainData.values


    # In[227]:


    #function execution
    splitEnergy(2012,2013)
```

Q2

```
#!/usr/bin/env python
# coding: utf-8

# ### import statements

# In[18]:


import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from pprint import pprint


# ### Question 3 Regression

# #### 3.1 & 3.2

# In[19]:


# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Split the data into training/testing sets
# Split the targets into training/testing sets

diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test = 
train_test_split(diabetes_X, diabetes_y, test_size=0.20, random_state=42)


# #### 3.3

# In[20]:
```

```
# Create linear regression object
linreg = linear_model.LinearRegression()

# Train the model using the training sets
linreg.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = linreg.predict(diabetes_X_test)


# In[21]:


# The coefficients
print('Coefficients: \n', linreg.coef_)

# The coefficient of determination: 1 is perfect prediction
print('\nR-squared score: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))


# #### 3.4

# In[22]:


#Using 10-fold cross validation to fit and validate linear regression models on
the whole data set. Printing the scores for each validation.

scores = cross_val_score(linreg, diabetes_X_train, diabetes_y_train, cv=10)
count = 1
for i in scores:
    print("Cross Validation Score for fold ", count, "is: ", i)
    count = count + 1

print("\nMean Accuracy: %0.2f with STD DEVIATION:  %0.2f" %
(np.mean(np.abs(scores)), scores.std() * 2))


# #### 3.5
```

```
# In[26]:


# Use sklearn to create RandomForestRegressor model, and fit the training data
into it.

rforest = RandomForestRegressor(n_estimators=1000, max_depth=None,
min_samples_split=2, random_state=1)
rforest.fit(diabetes_X_train, diabetes_y_train)

diabetes_y_pred = rforest.predict(diabetes_X_test)
#calculating the r square score
print('\nR-squared score: %.2f'% r2_score(diabetes_y_test, diabetes_y_pred))

#calculating root mean sq error
print("Root mean squared test error =
{0}".format(np.sqrt(np.mean((rforest.predict(diabetes_X_test) -
diabetes_y_test)**2))))


# #### 3.6

# In[24]:


#Using Grid Search to find the optimal hyper-parameters

#setup randomforest
rforest = RandomForestRegressor(n_estimators=1000, max_depth=None,
min_samples_split=2, random_state=1)

#setup and train GridSearchCV
clf = GridSearchCV(rforest, {'max_depth': [None, 7, 4],'min_samples_split': [2,
10, 20]}, cv=10)
model = clf.fit(diabetes_X_train, diabetes_y_train)

# print optimal hyper-parameters
pprint(model.best_estimator_.get_params())


# In[25]:
```

```
#print R-squared score
diabetes_y_pred = model.predict(diabetes_X_test)
print('\nR-squared score: %.2f'% r2_score(diabetes_y_test, diabetes_y_pred))
```