CS 535 Fall 2018 (project report)

Introduction:

Time series prediction is an important aspect of Data Mining, as it can provide valuable information about the future sales and customer behavior. The companies can leverage this tool to strategically allocate their resources and identify potential customers.

Objective:

In this project, we have been given sales data of 100 key products over 118 days. It also includes the details of customers including their ages. We have been tasked to forecast the individual sales of 100 key products and combined sales of 100 products for next 29 days.

Method:

I have developed a python based model to forecast the time series in order to complete this project. The process of model development is divided into three parts:

- Identification: Since, the sales data demonstrate a combination of oscillations and upward and downward trends, a robust model that can handle seasonality, non-stationary data is required for forecasting. I have gone through various resources including Wikipedia, reference book, and course material to identify SARIMAX as a suitable model for this project. I have also investigated ARIMA model, however, it was challenging to converge and obtain reasonable forecasting results.
- Training and testing: The training is the part where I identify the suitable parameters for the SARIMAX model. Initially, I have divided the data into two sets: training and testing. The first 90% of sales data is used for training and rest are used to testing of the model. I have employed a grid search approach to identify the suitable parameters for the model. In this grid search approach, I pick a combination of (p,d,q) and build the respective model to fit the training data. Then the model is used to forecast for the testing data and an error residual is calculated between forecasted and testing data. A looping is performed over (p,d,q), and the (p,d,q) associated with the minimum residual between forecasted and actual testing data is picked as the optimum model parameter for the give dataset.
- Forecasting: The model parameter obtained from previous step is used for forecasting 29 days in future. The approach is iteratively applied to all 101 data sets to obtain their forecast.

Program:

The details of implementation is provided below:

1) Loading of data file

```
# Import key product IDs and product distribution training set
key_product_IDs = pd.read_table("key_product_IDs.txt", header=None)
product_distribution_training_set = pd.read_table("product_distribution_training_set.txt",
header=None)
product_distribution_training_set = product_distribution_training_set.T

product_distribution_training_set =
product_distribution_training_set.drop(product_distribution_training_set.index[0])
product_distribution_training_set.index = pd.date_range(start='11/1/2018', periods=118)
product_distribution_training_set.index =
pd.to_datetime(product_distribution_training_set.index)

# Make entry for total sales numbers
key_product_IDs.columns = ["key_product_ID"]
```

```
key_product_IDs = key_product_IDs.set_value(len(key_product_IDs), 'key_product_ID', 0)
product_distribution_training_set['0'] =
product_distribution_training_set[product_distribution_training_set.columns].sum(axis=1)
```

2) Extraction of training and testing data sets

```
for iproduct in range(101):

  print(iproduct)
  trainStartIndex = 0
  trainEndIndex = np.floor(len(product_distribution_training_set.index)*0.9).astype(int)
  testStartIndex = trainEndIndex
  testEndIndex = len(product_distribution_training_set.index)

  sampleData= product_distribution_training_set.iloc[trainStartIndex:testEndIndex, iproduct]
  trainData = product_distribution_training_set.iloc[trainStartIndex:trainEndIndex, iproduct]
  testData = product_distribution_training_set.iloc[testStartIndex:testEndIndex, iproduct]
```

3) Stationary check

```
def test_stationarity(timeseries):

  #Determing rolling statistics
  rolmean = timeseries.rolling(window=7).mean()
  rolstd = timeseries.rolling(window=7).std()

  #Plot rolling statistics:
  plt.rcParams["figure.figsize"] = [11, 8]
  plt.plot(timeseries, '.-', color='blue', label='Original', linewidth=1.0)
  plt.plot(rolmean, '.-', color='black', label='Rolling Mean', linewidth=1.0)
  plt.plot(rolstd, '.-', color='red', label = 'Rolling Std', linewidth=1.0)
  plt.xticks(rotation='vertical')
  plt.legend(loc='best')
  plt.title('Rolling Mean & Standard Deviation')
  plt.show(block=False)

  #Perform Dickey-Fuller test:
  print('Results of Dickey-Fuller Test:')
  dftest = adfuller(timeseries, autolag='AIC')
  dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of
Observations Used'])
  for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
  print(dfoutput)
```

4) Seasonality check

```
def check_seasonality(data):
  from statsmodels.tsa.seasonal import seasonal_decompose
  decomposition = seasonal_decompose(data)

  trend = decomposition.trend
  seasonal = decomposition.seasonal
  residual = decomposition.resid
  plt.rcParams["figure.figsize"] = [11, 22]
  plt.subplot(411)
  plt.plot(data, '.-', color='blue', label='Original', linewidth=1.0)
  plt.xticks(rotation='vertical')
  plt.legend(loc='best')
  plt.subplot(412)
  plt.plot(trend, '.-', color='black', label='Trend', linewidth=1.0)
  plt.xticks(rotation='vertical')
  plt.legend(loc='best')
  plt.subplot(413)
  plt.plot(seasonal, '.-', color='red',label='Seasonality', linewidth=1.0)
  plt.xticks(rotation='vertical')
  plt.legend(loc='best')
  plt.subplot(414)
  plt.plot(residual, '.-', color='green', label='Residuals', linewidth=1.0)
  plt.xticks(rotation='vertical')
  plt.legend(loc='best')
  plt.tight_layout()
```

5) Grid search approach to obtain (p,d,q)

```
def findModelParameters(trainData,testData):
    min = 100000

    for i in range(1,5): # looping over p
        for j in range(1,5): # looping over q
            for k in range (0,2): # looping over d
                if i == 0 and j == 0:
                    continue
                else :
                    stepwise_model = SARIMAX(trainData, order=(i,k,j),
seasonal_order=(2,0,0,7), enforce_stationarity=False, enforce_invertibility=False)
                    try:
                        results_SARIMAX = stepwise_model.fit(disp=-1)
                    except:
                        continue
                    future_forecast = results_SARIMAX.forecast(len(testData)).astype(int)
                    RSS = (sum((future_forecast-testData)**2))
                    print('ARIMA(%d,0,%d) ==> AIC :: %2f, BIC :: %2f, RSS :: %2f' %
(i,j,results_SARIMAX.aic, results_SARIMAX.bic, RSS))
                    if (RSS < min):
                        min = RSS
                        p = i
                        q = j
                        d = k
    print('\n\nselected model :: SARIMAX(%d,%d,%d)' % (p,d,q))
    return (p,d,q)
```

6) Forecasting for 29 days

```
   #find most suitable model for the prediction using
  p,d,q = findModelParameters(sampleData,testData)
  print('\n\nselected model :: ARIMA(%d,%d,%d)' % (p,d,q))

  stepwise_model = SARIMAX(sampleData, order=(p,d,q), seasonal_order=(2,0,0,7),
enforce_stationarity=False, enforce_invertibility=False)
  results_SARIMAX = stepwise_model.fit(disp=-1)
  future_forecast = results_SARIMAX.forecast(29).astype(int)
  future_forecast = future_forecast.astype(int)
```

7) Writing of forecast data

```
  if writeflag == 0 :
    predicted_combined = future_forecast
    writeflag = 1
  else:
    predicted_combined = pd.concat([predicted_combined, future_forecast], axis=1)

#adjustment of data to write in the required format
```

```
sum_col = predicted_combined[0]
predicted_combined.drop(labels=[0], axis=1,inplace = True)
predicted_combined.insert(0, 0, sum_col)

predicted_combined = predicted_combined.T
predicted_combined.to_csv('predicted.csv', sep=' ', encoding='utf-8', header=False)
```

Guidelines to run the program:

The code is developed using python 3 libraries and will require **pandas, numpy, statsmodel, and matlibplot** to successfully run the program and visualize the results. The following commands should be used to run the program.

```
$python submission.py
```

Results:

I am including the description of two product IDs: 0 and 1, but the explanation can be extended to other product IDs.

- Combined sales
- Single product sale

Summary