sales_predictions_UnsupervisedMatrixMultiplication

June 10, 2022

1 Sales Predictions using Time Series Data

- https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data

1.1 Overview of Problem

"You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge." *(src: competition page)*

1.2 Imports

[45]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import NMF
```

1.3 Load Data

- The data provided from the Kaggle competition was edited and saved
- This notebook will load the updated file and continue from there

[2]:
```
# Load the data
```

[3]:
```
dfSales = pd.read_csv("dfShopItemsFull.csv")
```

[4]:
```
# Quick Stats on the data
dfSales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44443440 entries, 0 to 44443439
Data columns (total 4 columns):
 #   Column          Dtype
---  ------          -----
 0   date_block_num  int64
 1   shop_id         int64
 2   item_id         int64
 3   item_cnt_month  float64
dtypes: float64(1), int64(3)
memory usage: 1.3 GB
```

[5]:
```
dfSales.isnull().sum()
```

[5]:
```
date_block_num    0
shop_id           0
item_id           0
item_cnt_month    0
dtype: int64
```

[6]:
```
dfSales.head()
```

[6]:
```
   date_block_num  shop_id  item_id  item_cnt_month
0               0        0       32             6.0
1               0        0       33             3.0
2               0        0       35             1.0
3               0        0       43             1.0
4               0        0       51             2.0
```

1.3.1 Matrix Factorization

- The idea in the matrix multiplication model is to use the matrix we created in the EDA notebook and factor it
- This will help us find latent factors that we can use to predict missing data

[8]:
```
dfTrain = dfSales.copy() # Make a copy that we will mess up
month_size = dfTrain[dfTrain['date_block_num'].astype(int)==1].shape[0]

trainy = dfTrain.iloc[month_size:,:]['item_cnt_month'].reset_index().
 ↪drop(['index'],axis=1)
```

[10]:
```
dfTrain['next_month'] = trainy
```

[13]:
```
dfTrain = dfTrain.fillna(0)
```

[14]:
```
# If I dont have enough memory, I will just change dfSales
dfTest = dfSales[dfSales['date_block_num'] == 33]
```

[8]:
```
# Zero out the values for the last month and see what happens
#dfTrain.loc[dfSales['date_block_num'] == 33,'item_cnt_month'] = 0
```

[15]:
```
dfTest
```

[15]:
```
          date_block_num  shop_id  item_id  item_cnt_month
43136280              33        0       32             0.0
43136281              33        0       33             0.0
```

```
43136282                     33        0       35              0.0
43136283                     33        0       43              0.0
43136284                     33        0       51              0.0
…                         …      …      …             …
44443435                     33       36    12733              0.0
44443436                     33       36    13092              0.0
44443437                     33       36    16797              0.0
44443438                     33       36    18060              0.0
44443439                     33       36    15925              0.0

[1307160 rows x 4 columns]
```

[16]:

```
# There are negative values for some months.
# How should we handle this
dfTrain[dfTrain['item_cnt_month'] < 0].describe()
```

[16]:

```
       date_block_num     shop_id       item_id  item_cnt_month  next_month
count      912.000000  912.000000    912.000000      912.000000  912.000000
mean        14.121711   28.736842   9752.044956       -1.081140    0.383772
std          9.364640   17.095870   6235.720134        0.853736    0.955474
min          0.000000    2.000000     31.000000      -22.000000   -2.000000
25%          6.000000   12.000000   4351.750000       -1.000000    0.000000
50%         13.000000   27.000000   8106.500000       -1.000000    0.000000
75%         22.000000   44.000000  14503.250000       -1.000000    0.000000
max         33.000000   59.000000  22164.000000       -1.000000    9.000000
```

[21]:

```
# I assume it will be negative if there are returns.
# We could just right shift by the min, but I think it is ok to just 0 these out
dfTrain.loc[dfTrain['item_cnt_month'] < 0, 'item_cnt_month'] = 0
dfTrain.loc[dfTrain['next_month'] < 0, 'next_month'] = 0
```

[23]:

```
model_NMF = NMF(n_components=4,max_iter=700,init='nndsvda') # Basic model␣
  ↪before tweaking
nmf_fit = model_NMF.fit_transform(dfTrain)
```

[24]:

```
nmf_fit
```

[24]:

```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.82200147e-03],
       [1.11424001e-05, 9.13996588e-08, 0.00000000e+00, 4.95702248e-03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.27406353e-03],
       …,
       [7.41809603e-03, 8.99911858e-06, 0.00000000e+00, 2.52070295e+00],
       [8.52856697e-03, 7.12171543e-06, 0.00000000e+00, 2.70946389e+00],
       [6.65140509e-03, 1.02953146e-05, 0.00000000e+00, 2.39037870e+00]])
```

[25]:

```
C = model_NMF.components_
R_estimated = np.dot(nmf_fit, C)
```

[30]:
```
dfTrain.shape
```

[30]:
```
(44443440, 5)
```

[38]:
```
y_hat = R_estimated[:,4].astype(int)
```

[39]:
```
y_hat
```

[39]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[44]:
```
sum(y_hat)
# Just to spot check that it isnt all 0
```

[44]:
```
1091696
```

[42]:
```
y_hat.shape
```

[42]:
```
(44443440,)
```

[43]:
```
trainy.shape
```

[43]:
```
(43136280, 1)
```

[47]:
```
# Checking against training data
# Need to submit to Kaggle competition to get test data results
mean_squared_error(trainy, y_hat[:trainy.shape[0]])
```

[47]:
```
0.945051033607905
```

1.3.2 Format Data to Post to Kaggle

[80]:
```
# Using original sales matrix cause matrix mult may have messed up some data in␣
 ↪dfTrain
dfSales['yhat'] = y_hat
```

[88]:
```
dfSales['next_month'] = dfTrain['next_month']
```

[93]:
```
dfPredict34 = dfSales.iloc[-month_size:]
```

[94]:
```
dfPredict34['yhat'].sum()
```

[94]:
```
18575
```

[107]:
```
dfTest = pd.read_csv("../input/future-sales/test.csv")
```

[108]:

```
# I am going to merge with test so I want to make shop_id and item_id the␣
 ↪indices
dfPredict34.set_index(['shop_id','item_id'],inplace=True)
dfTest.set_index(['shop_id','item_id'],inplace=True)
```

[109]:

```
dfTest['item_cnt_month']=dfPredict34['next_month']
```

[110]:

```
dfTest
```

[110]:

```
                     ID  item_cnt_month
shop_id item_id
5       5037          0             0.0
        5320          1             NaN
        5233          2             0.0
        5232          3             0.0
        5268          4             NaN
…                    …             …
45      18454    214195             0.0
        16188    214196             0.0
        15757    214197             0.0
        19648    214198             0.0
        969      214199             0.0

[214200 rows x 2 columns]
```

[111]:

```
# Put the indices back to normal
dfTest.reset_index(inplace=True)
dfPredict34.reset_index(inplace=True)
```

[115]:

```
# Fix some missing items
# We can improve on how we impute - FUTURE WORK
# Manual inspection shows that several missing items are similar to the next␣
 ↪item id over
# This is not always true but will use it for initial impute
# In many cases, it is the same game but on different platform
# A better impute would check the text string then compare with statistical␣
 ↪trends of the platform
# Is PS4 or Xbox more popular?

i = 0
for index,row in dfTest[dfTest['item_cnt_month'].isnull()].iterrows():
    item_id = row['item_id'].astype(int)
    # Try add one
    query_impute =␣
 ↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id +1 ) &␣
 ↪(dfPredict34['shop_id']==row['shop_id'])]
    while query_impute.shape[0] == 0: # Try  remove 1 until we have a match
```

```
        item_id -= 1
        query_impute =␣
 ↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id - 1 ) &␣
 ↪(dfPredict34['shop_id']==row['shop_id'])]
    # if it is a series then look at it and see what is up
    #print(type(query_impute))
    #if not isinstance(query_impute,np.float64):
    #    print("error",query_impute)
    #    break
    dfTest.loc[index,'item_cnt_month'] = float(query_impute)
```

[117]:

```
# Save CSV
dfTest[['ID','item_cnt_month']].to_csv("sample_submission_MM.csv",index=False)
```

1.4 Analysis and Results

- This method worked well using the training set
- We were able to obtain a good RMSE value when we evaluated predicted results against the training data
- The test data has an RMSE of 1.26
- This is a fair but not great RMSE
- We can improve this by adding some features based on clustering items and shops
- We discuss this more in the LSTM notebook