FM_Dataset_preparation

December 19, 2019

```
In [0]: import pandas as pd
        import numpy as np
        import random
        import matplotlib.pyplot as plt
        import time
        import sklearn
        from sklearn.model_selection import train_test_split
        from random import shuffle
        import seaborn as sns
```

```
In [0]: from google.colab import drive
        drive.mount('/content/drive',force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/drive
```

```
In [0]: path="/content/drive/My Drive/yelp_final_data/"
```

0.0.1 In this notebook, we prepare the large dataset to train the FM model, and also the segmented test set.

Read the data files we saved. We only want the restaurant that has been rated more than twice and the users that rated at least 5 times. This condition should already be satisfied when we prepared the data files, we just do it here again for double checking.

We also read the test set index we saved before.

```
In [0]: #start_time=time.time()
        review=pd.read_csv(path+'review.csv')
        del review['text_review']
        review['freq_business'] = review.groupby('business_id')['business_id'].transform('coun
        review2=review.loc[review['freq_business']>2]
        review2['freq_user'] = review2.groupby('user_id')['user_id'].transform('count')
        review3=review2.loc[review2['freq_user']>=5]
        review3=review3.reset_index()
        test_idx=pd.read_csv(path+'all_test_idx_df2.csv')
        test_idx=test_idx.rename({'0':'index'},axis=1)
        test=review3.loc[review3['index'].isin(test_idx['index'])]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/i
  """
```

Based on the selected test row index we can create train and test dataset.

```
In [0]: train=review3.loc[~review3['index'].isin(test_idx['index'])]
```

Read user and business table and join the tables on user_id and business_id.

```
In [0]: user=pd.read_csv(path+'user2.csv')
        business=pd.read_csv(path+'business.csv')
        business=business.rename(columns={"business_ids": "business_id"})

In [0]: train = pd.merge(train, user, on='user_id')
        test=pd.merge(test, user, on='user_id')

In [0]: train = pd.merge(train, business, on='business_id')
        test = pd.merge(test, business, on='business_id')

In [0]: train.head(2)

Out[0]:     index  ...                                              hours
        0       0  ...  {'Monday': '0:0-0:0', 'Tuesday': '0:0-0:0', 'W...
        1  186281  ...  {'Monday': '0:0-0:0', 'Tuesday': '0:0-0:0', 'W...

        [2 rows x 38 columns]
```

Select the features we want to use when training the model. We have tried different combination of features. However, for the succinctness of the notebook, we only demonstrate the final features we chose.

```
In [0]: train1=train[['user_id','business_id','average_stars','stars','city','state','rating_r
        test1=test[['user_id','business_id','average_stars','stars','city','state','rating_rev

In [0]: train1.head(2)

Out[0]:                   user_id             business_id  ...  state  rating_review
        0  hG7b0MtEbXx5QzbzE6C_VA  ujmEBvifdJM6h6RLv4wQIg  ...     NV            1.0
        1  hG7b0MtEbXx5QzbzE6C_VA  ujmEBvifdJM6h6RLv4wQIg  ...     NV            1.0

        [2 rows x 7 columns]
```

Renumber the user_id and business_id from 0.

```
In [0]: user_id_addresses = train1.user_id.unique()
        user_id_dict = dict(zip(user_id_addresses, range(len(user_id_addresses))))
        train1=train1.applymap(lambda s: user_id_dict.get(s) if s in user_id_dict else s)
        test1=test1.applymap(lambda s: user_id_dict.get(s) if s in user_id_dict else s)
```

```
In [0]: total_business_id=list(train1.business_id.unique())+list(test1.business_id.unique())
```

```
In [0]: business_id_dict = dict(zip(total_business_id, range(len(total_business_id))))
```

```
In [0]: # business_id_addresses = train1.business_id.unique()
        # business_id_dict = dict(zip(business_id_addresses, range(len(business_id_addresses))
        train1=train1.applymap(lambda s: business_id_dict.get(s) if s in business_id_dict else
        test1=test1.applymap(lambda s: business_id_dict.get(s) if s in business_id_dict else s
```

```
In [0]: train1.head(2)
```

```
Out[0]:    user_id  business_id  average_stars  stars       city state  rating_review
        0        0       151026            2.0    2.5  Las Vegas    NV            1.0
        1        0       151026            2.0    2.5  Las Vegas    NV            1.0
```

Save the train and test set as txt file.

```
In [0]: np.savetxt(path+'test1.txt', test1.values, fmt=['%d','%d','%d','%d','%s','%s','%d'])
```

```
In [0]: np.savetxt(path+'train1.txt', train1.values, fmt=['%d','%d','%d','%d','%s','%s','%d'])
```

0.0.2 Prepare segmented test dataset in user and business dimension

The reason we are creating separated test dataset is because that we want to test our models on different levels of users and businesses.

Same data preparation logic applies here, we perform the same procedure above on different classes of users and businesses we segmented before.

```
In [0]: review=pd.read_csv(path+'review.csv')
        del review['text_review']
        review['freq_business'] = review.groupby('business_id')['business_id'].transform('coun
        review2=review.loc[review['freq_business']>2]
        review2['freq_user'] = review2.groupby('user_id')['user_id'].transform('count')
        review3=review2.loc[review2['freq_user']>=5]
        review3=review3.reset_index()
        test_idx=pd.read_csv(path+'all_test_idx_df2.csv')
        test_idx=test_idx.rename({'0':'index'},axis=1)
        test=review3.loc[review3['index'].isin(test_idx['index'])]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/i
  """
```

Read the segmented userID/businessID we created before, get the corresponding rows from the test set according to the userID/businessID, join them with user information and business information table, take the feature columns we want, renumber the userid and businessid from 0, and save them as txt files.

```
In [0]: unpopular_userid=pd.read_csv(path+'unpopular_user_ID.csv')
        midpopular_userid=pd.read_csv(path+'midpopular_user_ID.csv')
        popular_userid=pd.read_csv(path+'popular_user_ID.csv')
```

```
In [0]: unpopular_user=test.loc[test['user_id'].isin(unpopular_userid['userId'])]
        midpopular_user=test.loc[test['user_id'].isin(midpopular_userid['userId'])]
        popular_user=test.loc[test['user_id'].isin(popular_userid['userId'])]
```

```
In [0]: user=pd.read_csv(path+'user2.csv')
        business=pd.read_csv(path+'business.csv')
        business=business.rename(columns={"business_ids": "business_id"})
```

```
In [0]: unpopular_user=pd.merge(unpopular_user, user, on='user_id')
        midpopular_user=pd.merge(midpopular_user, user, on='user_id')
        popular_user=pd.merge(popular_user, user, on='user_id')
```

```
In [0]: unpopular_user = pd.merge(unpopular_user, business, on='business_id')
        midpopular_user = pd.merge(midpopular_user, business, on='business_id')
        popular_user = pd.merge(popular_user, business, on='business_id')
```

```
In [0]: unpopular_user=unpopular_user[['user_id','business_id','average_stars','stars','city',
        midpopular_user=midpopular_user[['user_id','business_id','average_stars','stars','city
        popular_user=popular_user[['user_id','business_id','average_stars','stars','city','sta
```

```
In [0]: user_id_addresses = train1.user_id.unique()
        user_id_dict = dict(zip(user_id_addresses, range(len(user_id_addresses))))

        unpopular_user=unpopular_user.applymap(lambda s: user_id_dict.get(s) if s in user_id_di
        midpopular_user=midpopular_user.applymap(lambda s: user_id_dict.get(s) if s in user_id_
        popular_user=popular_user.applymap(lambda s: user_id_dict.get(s) if s in user_id_dict e

        total_business_id=list(train1.business_id.unique())+list(test1.business_id.unique())
        business_id_dict = dict(zip(total_business_id, range(len(total_business_id))))
        unpopular_user=unpopular_user.applymap(lambda s: business_id_dict.get(s) if s in busine
        midpopular_user=midpopular_user.applymap(lambda s: business_id_dict.get(s) if s in busi
        popular_user=popular_user.applymap(lambda s: business_id_dict.get(s) if s in business_i
```

```
In [0]: np.savetxt(path+'unpopular_user.txt', unpopular_user.values, fmt=['%d','%d','%d','%d',
        np.savetxt(path+'midpopular_user.txt', midpopular_user.values, fmt=['%d','%d','%d','%d
        np.savetxt(path+'popular_user.txt', popular_user.values, fmt=['%d','%d','%d','%d','%s'
```

```
In [0]: unpopular_businessid=pd.read_csv(path+'unpopular_business_ID.csv')
        midpopular_businessid=pd.read_csv(path+'midpopular_business_ID.csv')
        popular_businessid=pd.read_csv(path+'popular_business_ID.csv')
```

```
In [0]: unpopular_business=test.loc[test['business_id'].isin(unpopular_businessid['businessId']
        midpopular_business=test.loc[test['business_id'].isin(midpopular_businessid['businessId
        popular_business=test.loc[test['business_id'].isin(popular_businessid['businessId'])]

In [0]: unpopular_business=pd.merge(unpopular_business, user, on='user_id')
        midpopular_business=pd.merge(midpopular_business, user, on='user_id')
        popular_business=pd.merge(popular_business, user, on='user_id')

In [0]: unpopular_business = pd.merge(unpopular_business, business, on='business_id')
        midpopular_business = pd.merge(midpopular_business, business, on='business_id')
        popular_business= pd.merge(popular_business, business, on='business_id')

In [0]: unpopular_business=unpopular_business[['user_id','business_id','average_stars','stars'
        midpopular_business=midpopular_business[['user_id','business_id','average_stars','stars
        popular_business=popular_business[['user_id','business_id','average_stars','stars','ci

In [0]: # user_id_addresses = train1.user_id.unique()
        # user_id_dict = dict(zip(user_id_addresses, range(len(user_id_addresses))))

        unpopular_business=unpopular_business.applymap(lambda s: user_id_dict.get(s) if s in us
        midpopular_business=midpopular_business.applymap(lambda s: user_id_dict.get(s) if s in
        popular_business=popular_business.applymap(lambda s: user_id_dict.get(s) if s in user_i

        # total_business_id=list(train1.business_id.unique())+list(test1.business_id.unique())
        # business_id_dict = dict(zip(total_business_id, range(len(total_business_id))))
        unpopular_business=unpopular_business.applymap(lambda s: business_id_dict.get(s) if s i
        midpopular_business=midpopular_business.applymap(lambda s: business_id_dict.get(s) if s
        popular_business=popular_business.applymap(lambda s: business_id_dict.get(s) if s in bu

In [0]: np.savetxt(path+'unpopular_business.txt', unpopular_business.values, fmt=['%d','%d','%
        np.savetxt(path+'midpopular_business.txt', midpopular_business.values, fmt=['%d','%d',
        np.savetxt(path+'popular_business.txt', popular_business.values, fmt=['%d','%d','%d','
```