Factorization_Machine

December 19, 2019

```
In [0]: !pip install git+https://github.com/coreylynch/pyFM
```

```
Collecting git+https://github.com/coreylynch/pyFM
  Cloning https://github.com/coreylynch/pyFM to /tmp/pip-req-build-btaxvm_u
  Running command git clone -q https://github.com/coreylynch/pyFM /tmp/pip-req-build-btaxvm_u
Building wheels for collected packages: pyfm
  Building wheel for pyfm (setup.py) ... done
  Created wheel for pyfm: filename=pyfm-0.0.0-cp36-cp36m-linux_x86_64.whl size=220628 sha256=0
  Stored in directory: /tmp/pip-ephem-wheel-cache-ani4rfpo/wheels/3b/d9/ef/1b148c527d393446328
Successfully built pyfm
Installing collected packages: pyfm
Successfully installed pyfm-0.0.0
```

```
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
```

0.0.1 This notebook is for training and testing the Factorization Model.

There are three parts: 1. Train and test on the large dataset. 2. Test on the segmented test set in which the large test set is segmented in user/business dimension. 3. Train and test on the small dataset.

```
In [0]: import numpy as np
        from pyfm import pylibfm
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import mean_absolute_error
```

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

Check the data.

```
In [0]: i=0
        with open(path+'train1.txt') as f:
                for line in f:
                  lst=line.split()
                  city=" ".join(lst[4:-2])
                  print(lst)
                  print(city)
                  i+=1
                  if i==10:
                    break
```

```
['0', '151026', '2', '2', 'Las', 'Vegas', 'NV', '1']
Las Vegas
['0', '151026', '2', '2', 'Las', 'Vegas', 'NV', '1']
Las Vegas
['1', '151026', '3', '2', 'Las', 'Vegas', 'NV', '3']
Las Vegas
['2', '151026', '3', '2', 'Las', 'Vegas', 'NV', '5']
Las Vegas
['2', '151026', '3', '2', 'Las', 'Vegas', 'NV', '4']
Las Vegas
['3', '151026', '3', '2', 'Las', 'Vegas', 'NV', '4']
Las Vegas
['4', '151026', '3', '2', 'Las', 'Vegas', 'NV', '3']
Las Vegas
['5', '151026', '3', '2', 'Las', 'Vegas', 'NV', '2']
Las Vegas
['6', '151026', '3', '2', 'Las', 'Vegas', 'NV', '4']
Las Vegas
['7', '151026', '3', '2', 'Las', 'Vegas', 'NV', '4']
Las Vegas
```

```
In [0]: def loadData(filename,path):
            data = []
            y = []
            users=set()
            items=set()
            #i=0
            with open(path+filename) as f:
                for line in f:
                    #print(i)
                    lst=line.split()
                    #if len(lst)>10:
                    city=" ".join(lst[4:-2])
                    # else:
                    #   city=lst[4]
                    # data.append({ "user_id": lst[0], "movie_id": lst[1],
                    #              'avg_star':int(lst[2]),'star':int(lst[3]),'city':city
                    #              })
                    data.append({ "user_id": lst[0], "movie_id": lst[1],
                                 'avg_star':int(lst[2]),'star':int(lst[3]),'city':city,'state'

                                 })
                    rating=lst[-1]
                    y.append(float(rating))
                    user=lst[0]
                    movieid=lst[1]
                    users.add(user)
                    items.add(movieid)
                    #i+=1

            return (data, np.array(y), users, items)
```

0.0.2 Train and test on larger dataset.

Read the txt file and save the features into the form we want.

```
In [0]: (train_data, y_train, train_users, train_items) = loadData(filename='train1.txt',path=p
        (test_data, y_test, test_users, test_items) = loadData(filename='test1.txt',path=path)
```

Transform the data into the form we want, such as one-hot-encoding user_id, business_id, etc.

```
In [0]: v = DictVectorizer()
        X_train = v.fit_transform(train_data)
        X_test = v.transform(test_data)
```

```
In [0]: X_train
```

```
Out[0]: <3618100x430652 sparse matrix of type '<class 'numpy.float64'>'
                with 21708600 stored elements in Compressed Sparse Row format>
```

Build and train a Factorization Machine

```
In [0]: fm = pylibfm.FM(num_factors=20, num_iter=15, verbose=True, task="regression", initial_l
```

```
In [0]: fm.fit(X_train,y_train)
```

```
Creating validation dataset of 0.01 of training for adaptive regularization
-- Epoch 1
Training MSE: 0.65979
-- Epoch 2
Training MSE: 0.65605
-- Epoch 3
Training MSE: 0.64930
-- Epoch 4
Training MSE: 0.64133
```

```
-- Epoch 5
Training MSE: 0.63601
-- Epoch 6
Training MSE: 0.63099
-- Epoch 7
Training MSE: 0.62687
-- Epoch 8
Training MSE: 0.62410
-- Epoch 9
Training MSE: 0.61980
-- Epoch 10
Training MSE: 0.61793
-- Epoch 11
Training MSE: 0.61566
-- Epoch 12
Training MSE: 0.61438
-- Epoch 13
Training MSE: 0.61193
-- Epoch 14
Training MSE: 0.61230
-- Epoch 15
Training MSE: 0.61050
```

```
In [0]: preds = fm.predict(X_test)
        print("FM RMSE: %.4f" % np.sqrt(mean_squared_error(y_test,preds)))
        print("FM MAE: %.4f" % mean_absolute_error(y_test,preds))
```

```
FM RMSE: 1.2154
FM MAE: 0.9757
```

Get the RSME and MAE of the last rating of each user.

```
In [0]: def last_rating(preds,y_test):
          #rmse=[]
          pred=[]
          true=[]

          for i in range(0,len(preds),3):
            pred.append(preds[i+2])
            true.append(y_test[i+2])
            #print(pred,true)
          rmse=np.sqrt(mean_squared_error(true,pred))
          mae=mean_absolute_error(y_test,preds)


          return rmse,mae
        rmse,mae=last_rating(preds,y_test)
        print('RMSE of last rating of each user',rmse)
        print('MAE of last rating of each user',mae)
```

```
RMSE of last rating of each user 1.2101783304850455
MAE of last rating of each user 0.9229001312142616
```

0.0.3 Defination of well_recommended user:

Since there are three ratings for each user in the test set, if the predicted ranking of these three businesses is exactly the same as the true ranking of these businesses, then this user would be counted as a well_recommended user. The ranking is ordered by the rating of the reviews of user to business.

Calculate user coverage: number of well-recommended user/total number of unique users.

```
In [0]: def cov(preds,y_test):
          count=0
          for i in range(0,len(preds),3):
            pred_rank=np.argsort([preds[i],preds[i+1],preds[i+2]])
            true_rank=np.argsort([y_test[i],y_test[i+1],y_test[i+2]])
            if (pred_rank==true_rank).all():
              count+=1
          return count/281365

        print('user_coverage',cov(preds,y_test))
```

```
user_coverage 0.2887850301210172
```

0.0.4 Test on segmented user and business

```
In [0]: def last_rating(preds,y_test):
          #rmse=[]
          pred=[]
          true=[]

          for i in range(0,len(preds),3):
            pred.append(preds[i+2])
            true.append(y_test[i+2])
            #print(pred,true)
          rmse=np.sqrt(mean_squared_error(true,pred))
          mae=mean_absolute_error(y_test,preds)


          return rmse,mae
```

```
In [0]: def cov(preds,y_test,num_user):
          count=0
          for i in range(0,len(preds),3):
            pred_rank=np.argsort([preds[i],preds[i+1],preds[i+2]])
            true_rank=np.argsort([y_test[i],y_test[i+1],y_test[i+2]])
            if (pred_rank==true_rank).all():
              count+=1
          return count/num_user
```

```
In [0]: (unpopular_user, unpopular_user_y, _, _) = loadData(filename='unpopular_user.txt',path=
        (midpopular_user, midpopular_user_y, _, _) = loadData(filename='midpopular_user.txt',pa
        (popular_user, popular_user_y,_, _) = loadData(filename='popular_user.txt',path=path)
```

```
In [0]: unpopular_user_test = v.transform(unpopular_user)
        midpopular_user_test = v.transform(midpopular_user)
        popular_user_test = v.transform(popular_user)
```

```
In [0]: print("Unpopular user")
        preds = fm.predict(unpopular_user_test)
        print("unpopular user FM RMSE: %.4f" % np.sqrt(mean_squared_error(unpopular_user_y,pre
        print("unpopular user FM MAE: %.4f" % mean_absolute_error(unpopular_user_y,preds))
        rmse,mae=last_rating(preds,unpopular_user_y)
        print('RMSE of last rating of each user',rmse)
        print('MAE of last rating of each user',mae)
        print('user_coverage',cov(preds,unpopular_user_y,143289))

        print('\n')
        print("Midpopular user")
        preds = fm.predict(midpopular_user_test)
        print("mid-popular user FM RMSE: %.4f" % np.sqrt(mean_squared_error(midpopular_user_y,
        print("mid-popular user FM MAE: %.4f" % mean_absolute_error(midpopular_user_y,preds))
        rmse,mae=last_rating(preds,midpopular_user_y)
        print('RMSE of last rating of each user',rmse)
        print('MAE of last rating of each user',mae)
        print('user_coverage',cov(preds,midpopular_user_y,78897))

        print('\n')
        print("popular user")

        preds = fm.predict(popular_user_test)
        print("popular user FM RMSE: %.4f" % np.sqrt(mean_squared_error(popular_user_y,preds))
        print("popular user FM MAE: %.4f" % mean_absolute_error(popular_user_y,preds))
        rmse,mae=last_rating(preds,popular_user_y)
        print('RMSE of last rating of each user',rmse)
        print('MAE of last rating of each user',mae)
        print('user_coverage',cov(preds,popular_user_y,59179))
```

```
Unpopular user
unpopular user FM RMSE: 1.2579
unpopular user FM MAE: 1.0008
RMSE of last rating of each user 1.2601614953727724
MAE of last rating of each user 1.0008270050532322
```

```
user_coverage 0.29424449887988613


Midpopular user
mid-popular user FM RMSE: 1.2455
mid-popular user FM MAE: 0.9919
RMSE of last rating of each user 1.2465255779766422
MAE of last rating of each user 0.9918964486591396
user_coverage 0.2870197852896815


popular user
popular user FM RMSE: 1.1982
popular user FM MAE: 0.9435
RMSE of last rating of each user 1.1976544833540301
MAE of last rating of each user 0.9435061460703181
user_coverage 0.2899339292654489
```

```
In [0]: (unpopular_business, unpopular_business_y, _, _) = loadData(filename='unpopular_busines
        (midpopular_business, midpopular_business_y, _, _) = loadData(filename='midpopular_busi
        (popular_business, popular_business_y,_, _) = loadData(filename='popular_business.txt'
```

```
In [0]: unpopular_business_test = v.transform(unpopular_business)
        midpopular_business_test = v.transform(midpopular_business)
        popular_business_test = v.transform(popular_business)
```

```
In [0]: print("Unpopular business")
        preds = fm.predict(unpopular_business_test)
        print("unpopular business FM RMSE: %.4f" % np.sqrt(mean_squared_error(unpopular_busines
        print("unpopular business FM MAE: %.4f" % mean_absolute_error(unpopular_business_y,pred


        print('\n')
        print("Midpopular business")
        preds = fm.predict(midpopular_business_test)
        print("mid-popular business FM RMSE: %.4f" % np.sqrt(mean_squared_error(midpopular_busi
        print("mid-popular business FM MAE: %.4f" % mean_absolute_error(midpopular_business_y,p



        print('\n')
        print("popular business")

        preds = fm.predict(popular_business_test)
        print("popular business FM RMSE: %.4f" % np.sqrt(mean_squared_error(popular_business_y
        print("popular business FM MAE: %.4f" % mean_absolute_error(popular_business_y,preds))
```

```
Unpopular business
unpopular business FM RMSE: 1.3176
unpopular business FM MAE: 1.0623


Midpopular business
mid-popular business FM RMSE: 1.3216
mid-popular business FM MAE: 1.0665


popular business
popular business FM RMSE: 1.2119
popular business FM MAE: 0.9571
```

0.0.5 Train and test on smaller dataset

```
In [0]: (train_data_small, y_train_small, train_users, train_items) = loadData(filename='traini
        (test_data_small, y_test_small, test_users, test_items) = loadData(filename='test1_smal
```

```
In [0]: v = DictVectorizer()
        X_train_s = v.fit_transform(train_data_small)
        X_test_s = v.transform(test_data_small)
```

```
In [0]: X_train_s
```

```
Out[0]: <258881x93001 sparse matrix of type '<class 'numpy.float64'>'
                with 1553286 stored elements in Compressed Sparse Row format>
```

```
In [0]: # Build and train a Factorization Machine
        fm = pylibfm.FM(num_factors=20, num_iter=15, verbose=True, task="regression", initial_
```

```
In [0]: fm.fit(X_train_s,y_train_small)
```

```
Creating validation dataset of 0.01 of training for adaptive regularization
-- Epoch 1
Training MSE: 0.66161
-- Epoch 2
Training MSE: 0.64962
-- Epoch 3
Training MSE: 0.64141
-- Epoch 4
Training MSE: 0.63436
-- Epoch 5
Training MSE: 0.62833
-- Epoch 6
Training MSE: 0.62248
-- Epoch 7
Training MSE: 0.61720
-- Epoch 8
Training MSE: 0.61232
```

```
-- Epoch 9
Training MSE: 0.60754
-- Epoch 10
Training MSE: 0.60306
-- Epoch 11
Training MSE: 0.59876
-- Epoch 12
Training MSE: 0.59482
-- Epoch 13
Training MSE: 0.59083
-- Epoch 14
Training MSE: 0.58707
-- Epoch 15
Training MSE: 0.58339
```

```
In [0]: preds_s = fm.predict(X_test_s)
        print("FM RMSE: %.4f" % np.sqrt(mean_squared_error(y_test_small,preds_s)))
        print("FM MAE: %.4f" % mean_absolute_error(y_test_small,preds_s))
```

```
FM RMSE: 1.2138
FM MAE: 0.9428
```

```
In [0]: def last_rating(preds,y_test):
          #rmse=[]
          pred=[]
          true=[]

          for i in range(0,len(preds),3):
            pred.append(preds[i+2])
            true.append(y_test[i+2])
            #print(pred,true)
          rmse=np.sqrt(mean_squared_error(true,pred))
          mae=mean_absolute_error(y_test,preds)


          return rmse,mae
        rmse,mae=last_rating(y_test_small,preds_s)
        print('RMSE of last rating of each user',rmse)
        print('MAE of last rating of each user',mae)
```

```
RMSE of last rating of each user 1.2110939133363248
MAE of last rating of each user 0.9628548533799136
```

```
In [0]: def cov(preds,y_test):
          count=0
          for i in range(0,len(preds),3):
```

```
    pred_rank=np.argsort([preds[i],preds[i+1],preds[i+2]])
    true_rank=np.argsort([y_test[i],y_test[i+1],y_test[i+2]])
    if (pred_rank==true_rank).all():
      count+=1
  return count/20000.0

print('user_coverage',cov(y_test_small,preds_s))
```

```
user_coverage 0.3216
```