ZHAOKAI XU (KAGGLE NAME: ZhaokaiXu)

CSE 158 Assignment 1

Prof Julian McAuley

Report of 158 Assignment 1

Task 1 Purchase Prediction

Purchase prediction Predict given **a (user,item) pair** from `pairs Purchase.txt' **whether the user purchased the item** (really, whether it was one of the products they reviewed). Accuracy will be measured in terms of the categorization accuracy (fraction of correct predictions). The test set has been constructed such that exactly 50% of the pairs correspond to purchased items and the other 50% do not.

I applied the idea of **decision tree** by using several predictors. Since the noise of the data is relatively high, I applied several filters/predictors as the node of the decision tree to obtain a final accuracy of 0.70 (ranking 121overall).

After trying different combinations of predictors and different features for prediction to build the decision tree, finally I **built a two-layers decision tree** as my overall predictor:

The first predictor is to filter data by popularity(frequency). By running data on the training and validation data set, I found **a good threshold is 20**. If the popularity of the item is over 20, I treated this item as a popular item and the user has a very high chance to buy it, so I marked it True. This is based on the common sense that a user is very likely to buy an item if the item is popular.

I did also try to build another predictor using **the average rating of an item**. My previous assumption is that if an item has a very low rating, then the probability to buy it should be low. However, after running the predictor on training and validation set, I found that predictor based on rating does not work well because there are very few items have a very low average rating.

The second predictor is to filter data by the category similarity of previous purchased history. Since the category (a set of words to describe an item) of an item is unique, and by adding all words related to the items that a user purchased before to the set of words for a user, we can calculate the **Jaccard similarity** of an item and a user using the two set of words. I found the best threshold is by setting the similarity to **0.17.**

To further improve the accuracy by loosing the predictor a little bit, I also calculated the **number of common words** in the user set of works and the set of words in the item by setting **the threshold of common words to be 5.**

Task 2 Category Prediction

Category prediction Predict the category of a business from a review. Five categories are used for this task, which can be seen in the baseline program, namely Women's, Men's, Girls', Boys', and Baby clothing. Performance will be measured in terms of the fraction of correct classi_cations.

I applied **sklearn Linear SVC to train five Linear SVC classifiers** and apply each of them to predict one categories, whether it predicts True OR False. If True predicted by one predictor, then we predict the item to be its category. **The linear SVC is running on the high dimension mapping of review Text.**

Before running Linear SVC, I mapped words into a high dimension space using some sklearn API from sklearn.feature_extraction.text including **CountVectorizer** and **TfidfTransformer**.

With **CountVectorizer,** I calculated the counts of words from reviewText.

With **TfidfTransformer,** I mapped words into a high dimension space to feed into a linear SVC.

I obtained a final accuracy of 0.85 (ranking 77 overall).

Overall Summary:

From the above two task, I learned some techniques to train classifiers for recommendation system such as calculating the Jaccard similarity and apply decision tree to further improve accuracy, and I obtained some skills to do data mining such as vector2word and Tfidf, which are popular used in data pre-processing in NLP (neural network processing)

158 Assignment 1 – Task 1

November 19, 2018

```
In [183]: import gzip
          from collections import defaultdict
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          from sklearn.utils import shuffle
          import random, heapq

In [184]: ##############################################################
          ########### reading data into a pandas dataframe #############
          data = []
          userRatings = defaultdict(list)

          def readGz(f):
              for l in gzip.open(f):
                  yield eval(l)
          #training and validation
          for l in readGz("train.json.gz"):
              data.append(l)

          print "total number of reviews: " , (len(data))
          df = pd.DataFrame.from_dict(data)


          validation_neg_pairs = set()
          for l in open("pairs_Purchase_neg.txt",'r'):
              if l.startswith("reviewerID"):
                  #header
                  continue
              u,i = l.strip().split('-')
              validation_neg_pairs.add((u,i))

          print(len(validation_neg_pairs))
```

```
total number of reviews:  200000
100000
```

0.1 overview

Since the noise of the data is relatively high, I applied several filters/predictors by common user experience of purchansing. By setting threshold based on statistically analysis, we should obtain a good overal predictor.

1 #### if an item is very poplular, it is highly possible to be purchased,

```
Here I found the best thershold is popularity over 20% then must be purchased
```

2 #### if the item category strings match most strings of a user purchased item category, then highly likely to be purchased.

```
Here I found the best threshold is  common strings > 5.
OR
the Jaccard similarity is larger than 0.17
```

```
In [208]: #setting up best f

          common_strings = 5

          Jac_threshold = 0.17

          Popularity_threshold = 20
```

```
In [209]: ####################################################################
          ############  filter one  category instead of categoryID ###########

          df_item_cat_text = df[['categories','itemID']]
          df_item_cat_text_uniq = df_item_cat_text.drop_duplicates(subset=['itemID'])
          df_item_cat_text_uniq.set_index('itemID',inplace=True)
```

```
In [210]: def flatten_cat_word_list(original_list):
              item_cat_set = set()
              for subcat in original_list:
                  for cat_str in subcat:
                      item_cat_set.add(cat_str)
              return item_cat_set
```

```
In [211]: # flatten the category list into 1d list of an item to set of strings
          d_item_to_cat_text={}   # itemID to set of string

          for idx,row in df_item_cat_text_uniq.iterrows():
              if idx not in d_item_to_cat_text:
                  d_item_to_cat_text[idx] = set()
              d_item_to_cat_text[idx] |= flatten_cat_word_list(row['categories'])
```

```
In [212]: # flatten the dictionary of userId to text strings
          d_user_to_cat_text={}   # userID to set of strings
          for idx, row in df.iterrows():
              userId,itemId = row['reviewerID'],row['itemID']

              if userId not in d_user_to_cat_text:
                  d_user_to_cat_text[userId] = set()

              d_user_to_cat_text[userId] |= d_item_to_cat_text[itemId]
```

```
In [213]: # check if the intersection of strings number is larger than 5
          def similarity(item_cat_text,user_cat_text):
              return  len(item_cat_text & user_cat_text) > common_strings \
              or float(len(item_cat_text & user_cat_text)) / len(item_cat_text | user_cat_text
```

```
In [214]: def predictor_category(itemId, userId):

              # check itemId or userId do not exist
              if itemId not in d_item_to_cat_text or userId not in d_user_to_cat_text:
                  return False

              # run similarity to predict
              return similarity(d_item_to_cat_text[itemId],d_user_to_cat_text[userId]) > Jac_t
```

```
In [215]: ####################################################################
          # ########### filter two ###########
          # filter the most popular items that have frequency higher than 20

          items_freq = df['itemID'].value_counts()
          plt_list = items_freq.value_counts().sort_index()
```

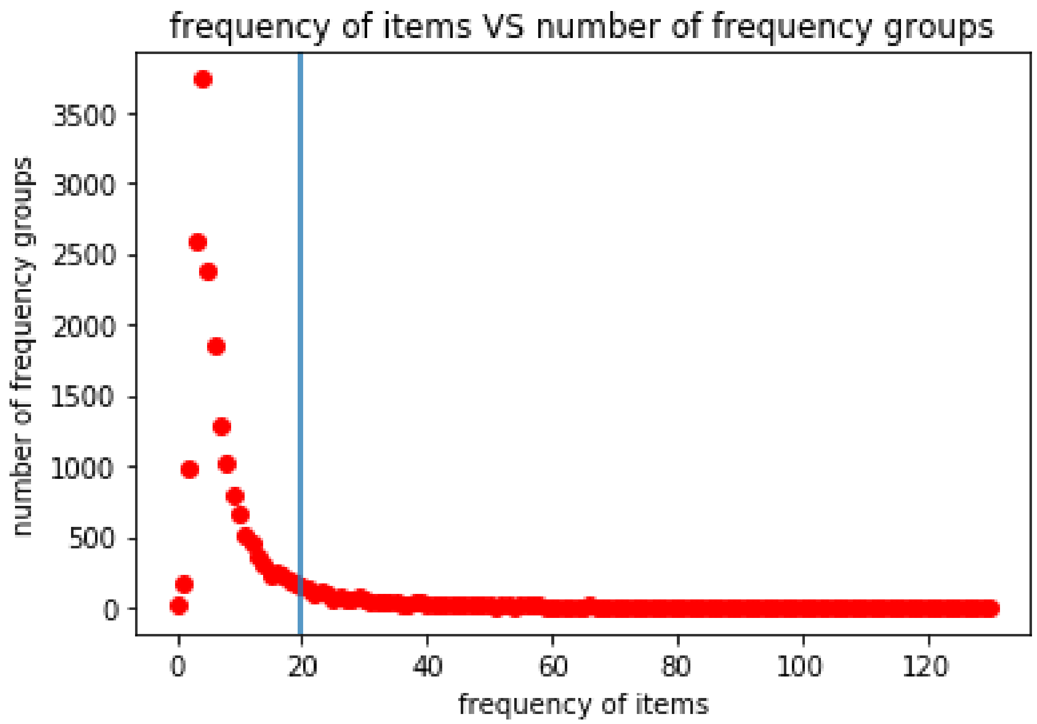
```
In [216]: list_num = len(plt_list)

          x = []
          y = []
          counter = 0
          for item in (plt_list):
              x.append(counter)
              counter += 1
              y.append(item)

          plt.title("frequency of items VS number of frequency groups")
          plt.plot(x,y,'ro')
          plt.xlabel("frequency of items")
          plt.ylabel("number of frequency groups")

          plt.axvline(x = Popularity_threshold)
```

```
Out[216]: <matplotlib.lines.Line2D at 0xa39aec3d0>
```



```
In [217]: # filter items by freq
          popular_items_freq = items_freq[items_freq > Popularity_threshold]
          print "popular items: ", len(popular_items_freq)
```

```
popular items:  1742
```

```
In [218]: ##################################################################
          #################### overall predictor ###########################

          predictions = open("predictions_Purchase.txt", 'w')
          predicted_true = 0
          predicted_false = 0
          lines = 0

          for l in open("pairs_Purchase.txt"):
              lines += 1
              if l.startswith("reviewerID"):
                  #header
                  predictions.write(l)
                  continue
              u,i = l.strip().split('-')

              # check the first predictor, if user purchased this category before
```

```
            if i in popular_items_freq:
                predictions.write(u + '-' + i + ",1\n")
                predicted_true += 1
            else:
                if predictor_category(i, u):
                    predictions.write(u + '-' + i + ",1\n")
                    predicted_true += 1
                else:
                    predictions.write(u + '-' + i + ",0\n")
                    predicted_false += 1

        print "number of line", lines
        print "predicted_true", predicted_true
        print "predicted_false: ", predicted_false

        predictions.close()
```

```
number of line 28001
predicted_true 14262
predicted_false:  13738
```

158 Assignment1 –Task2

November 19, 2018

```
In [1]: # basic lib
        import numpy as np
        import pandas as pd
        import gzip
        from collections import defaultdict
        import numpy as np

        # library for machine learning lib
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn import svm
        from sklearn.calibration import CalibratedClassifierCV

        def readGz(f):
            for l in gzip.open(f):
                yield eval(l)


        ###  reading the training data
        data = []
        userRatings = defaultdict(list)

        for l in readGz("train.json.gz"):
            data.append(l)

        print(len(data))

        ###  reading the test data

        data_test = []

        for l in readGz("test_Category.json.gz"):
            data_test.append(l)
        print(len(data_test))

        df2 = pd.DataFrame(data_test)
```

```
200000
```

```
14000
```

```
In [2]: #convert data to padas
        df = pd.DataFrame.from_dict(data)
```

```
In [11]: ### training linearing SVC

         X_train = df['reviewText']
         y_train = df['categoryID']

         count_vect = CountVectorizer()
         X_train_counts = count_vect.fit_transform(X_train)
         tf_transformer = TfidfTransformer().fit(X_train_counts)
         X_train_transformed = tf_transformer.transform(X_train_counts)


         linear_svc = svm.LinearSVC()
         clf = linear_svc.fit(X_train_transformed,y_train)

         calibrated_svc = CalibratedClassifierCV(base_estimator=linear_svc,
                                                 cv="prefit")

         calibrated_svc.fit(X_train_transformed,y_train)
```

```
Out[11]: CalibratedClassifierCV(base_estimator=LinearSVC(C=1.0, class_weight=None, dual=True, 
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.001,
              verbose=0),
                     cv='prefit', method='sigmoid')
```

```
In [12]: # function to predict test data

         def predict_single(row):
             reviewText = [row['reviewText']] # each document may contain multiple texts, so h
             p_count = count_vect.transform(reviewText)
             p_tfidf = tf_transformer.transform(p_count)
             pred_this = calibrated_svc.predict_proba(p_tfidf)
             return pred_this


         def pred2():
             pred_matrix = []

             for idx,row in df2.iterrows():

                 pred_this = predict_single(row)[0] # prediction is contained in a list
```

```
            pred_matrix.append(pred_this)
        return pred_matrix
```

```
In [13]: # function call to predict test set

         pred_matrix = pred2()

         df_pred2 = pd.DataFrame(pred_matrix)

         df_pred2['max_label'] = df_pred2.apply(lambda row: (np.argsort(-row[:5])[0]) ,  axis=

         prediction = list(df_pred2['max_label'])
```

```
In [14]: #  write the prediction to file

         predictions = open("predictions_Category.txt", 'w')
         predictions.write("reviewerID-reviewHash,category\n")
         for d,p in zip(data_test,prediction):
             predictions.write(d['reviewerID']+'-'+d['reviewHash']+','+str(p)+'\n')
         predictions.close()
```