CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

Task1: Visit Prediction
For visit prediction task, I tried two approaches. First, I will talk about the original approach I tried. This approach basically implements linear svm. I split and create the train/validation/test data as in homework 3. I use reviews 1-100,000 for training. I also create a validation set containing the 100,001 to 200,000 positive labels and another 100,000 randomly generated negative labels as in homework3. The difference is that in hw3, the training and validation datasets contain only the user/business pair without the 1/0 labels for svm training and validation. Therefore, I modified the hw3 training/validation sets to be a 3-tuple, (userID, businessID, label 0 or 1). For the feature vector, I have 3 features. User activity level, business activity level and user-business Pearson Correlation. To store user and business activity level, I use two defaultdict—userCount[userID] and businessCount[businessID]. The user-business Pearson Correlation is constructed by two functions—business-business Pearson and user-business Pearson. The business-to-business Pearson function simply takes two businessID as input and calculate the Pearson Correlation between them based on their ratings. The two rating arrays are accessed through defaultdict(list) userBusiness[businessID].

Below is the code for the business-business Pearson function.

```
def pearson_business(b1, b2):
  avg1 = np.mean([u['rating'] for u in userBusiness[b1]])
  avg2 = np.mean([u['rating'] for u in userBusiness[b2]])
  intersection = [{'b1': u1['rating'], 'b2': u2['rating']} for u1 in userBusiness[b1]
                  for u2 in userBusiness[b2] if u1['userID'] == u2['userID']]
  numer = (sum([(i['b1'] - avg1) * (i['b2'] - avg2) for i in intersection])) * 1.0
  denomin = (sum([(i['b1'] - avg1) ** 2 for i in intersection])
             * sum([(i['b2'] -avg2) ** 2 for i in intersection])) ** 0.5
  return 0.0 if denomin== 0.0 else numer/ denomin
```

Then for the value I want to include in my feature vector, I use the following function:

```
def pearson(user, business):
  value = 0.0
  if user not in businessUser or business not in userBusiness:
    return value
  for i in businessUser[user]:
    pearson_val = pearson_business(i['businessID'], business)
    if  pearson_val > 0.5:
        value = pearson_val
        return value
  return value
```

When processing data, we are inputting user/business pair to predict whether the user will visit the business. This functions loops through the input user's visited businesses and calculates the Pearson Correlation with the input business. The loop breaks once we encounter a value that is

CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

bigger than 0.5. This is the best parameter that I have tuned. Then we return this value and add to our feature vector. In the end, my feature vector looks like this:

```
def feature(d):
  f1=businessCount[d[1]]
  f2=userCount[d[0]]
  f3=pearson(d[0],d[1])
  feat = [1, f1, f2, f3]
  return feat
```

Then I build X_train and y_train to train linear svm:

```
x_train = [feature(d) for d in data_train]
y_train = [l[2] for l in data_train]
x_valid = [feature(d) for d in ub_validation]
y_valid = [d[2] for d in ub_validation]
clf=svm.LinearSVC(C=0.1)
clf.fit(x_train, y_train)
valid_predict = clf.predict(x_valid)
acc = [(x==y) for (x,y) in zip(valid_predict, y_valid)]
acc = sum(acc)*1.0/len(acc)
print(acc)
```

I use validation set to tune my parameter C for the svm. When C=0.1, this gives the best validation accuracy. I then use this model to predict user/business pairs in “pairs_Visit.txt" and submit my solution to Kaggle. The best score I receive for this model is 0.80275.

To further improve my ranking, I tried a better approach. Now I will talk about the second approach for the visit prediction task.
This approach simply implements collaborative filtering based on Jaccard Similarity. There is no use of svm.
I still use the two dictionaries userBusiness and businessUser to keep track of what businesses a user visited and what users visited a business. I create two functions to calculate business-business Jaccard Similarity and user-business similarity.
Below is the code for business-business Jaccard Similarity:

```
def jacard_business(b1,b2):

  set_1 = set([u['userID'] for u in userBusiness[b1]])
  set_2 = set([u['userID'] for u in userBusiness[b2]])
  n = float(len(set_1.intersection(set_2)))
  denominator = float(len(set_1) + len(set_2) - n)
  if denominator == 0.0: return 0.0
  return n /denominator
```

CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

My second function calculates user-business similarity and directly predict whether the input user would visit the input business. Below is the code:

```
def jacard(user, business):
  set_1 = userBusiness[user]
  set_2 = businessUser[business]
  p = 0.0007
  most_similar_value = 0.0
  for b in set_1:
    j = jacard_business(b,business)
    if j > most_similar_value:
        most_similar_value = j
  if most_similar_value > p:
    return 1
  else:
    return 0
```

First, I loop through all the businesses that the input user visited. During each iteration, I calculate the Jaccard similarity between this business and the input business by the above function I created. I keep track of the similarity values and get the maximum of them at the end of the loop. I then compare this value to a parameter p. The validation process is all about tuning the parameter p. When p=0.0007, I get the best accuracy. It also gives me the best submission score of 0.87620 on Kaggle.

In summary, I believe the reason why the first approach does not perform very well is that the Pearson values I calculate are based on ratings. However, ratings are not the strongest factor when predicting whether a user would visit a business. It is obviously more useful to study what businesses a user visited in the past and what is the similarity between those visited businesses and the input business we are trying to predict. Therefore, Jaccard Similarity is a better model.

CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

Task2: Category Prediction

For this task, I use just one approach and this approach gives me relatively good result. My approach is basically an improvement over the solution to hw3. This time, I simplify the process by using multiclass svm. First, I import OneVsRestClassifier and LinearSVC.

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
```

Then I filtered out those reviews from "train.json.gz" without categoryID. Then I split the filtered data in half. First half for training and second half for validation.

For feature vector design, I keep the 500 common words vector from hw3, and add similar vectors based on which words are more popular in one category compared to their overall frequency across all categories. To be specific, I modified the hw3 code as follow:

```
import string
from nltk.corpus import stopwords
stop_list = stopwords.words("english")
wordFreq = defaultdict(float)
wordFreqCat = {}
mydiff=defaultdict(list)
for c in range(10):
  wordFreqCat[c] = defaultdict(float)

punctuation = string.punctuation
for r in review_train:
  rText = ''.join([c for c in r['reviewText'].lower() if not c in punctuation])
  for w in rText.split():
    if w in stop_list:continue
    wordFreq[w] += 1.0 # Word frequency
    wordFreqCat[r['categoryID']][w] += 1.0 # Per-category frequency
totalWords = sum(wordFreq.values())
for w in wordFreq:
  wordFreq[w] /= totalWords

for c in range(10):
  totalWordsCat = sum(wordFreqCat[c].values())
  for w in wordFreqCat[c]:
    wordFreqCat[c][w] /= totalWordsCat

for c in range(10):
  diffs = []
  for w in wordFreq:
    diffs.append((wordFreqCat[c][w] - wordFreq[w], w))
  diffs.sort()
  diffs.reverse()
  topten = diffs[:250]
  for r in topten: mydiff[c].append(r[1])

most_common = [(wordFreq[w], w) for w in wordFreq]
most_common.sort()
most_common.reverse()
most_common_words = [w[1] for w in most_common[:500]]
```

CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

I find it helpful to remove the stopwords when determining the 500 most common words because by doing so the common words will be more representative. In addition, the feature vector in hw3 does not include the 10 words that are more frequent in one category compared to other categories. For this task, I include the most frequent words per category to my feature vector. First, I process and stored these frequent words by a dictionary named mydiff. The keys in this dictionary are basically the categoryIDs, i.e. 0-9. For each key, the value corresponds to a list of top frequent words within that category. Here, the number of top frequent words is not necessarily 10 as in hw3, rather, it is a parameter we need to tune. The number of most common words is also a parameter we need to tune.
After this step, I am able to create the feature vector. The code is below:

```
def feature(r):
  feat = [0] * len(commonWords)
  raw = ''.join([x for x in r['reviewText'].lower() if not x in string.punctuation])
  for word in raw.split():
    if word in word_set:
      feat[pos[word]] =1

  for i in range(10):
    sub_feat = [0] * 250
    common0= set(mydiff[i])
    pos0 = dict(zip(common0, range(250)))
    raw = ''.join([x for x in r['reviewText'].lower() if not x in string.punctuation])
    for word in raw.split():
      if word in common0:
        sub_feat[pos0[word]] =1

    feat+=sub_feat

  return feat
```

The first part is exactly the same as in hw3. The second part uses a loop to iterate through each category, and get the top frequent words in that category from dictionary mydiff. For each iteration, the vector created is the same type as the commonWords vector. In this code, I get the top 250 words from each category. I then create a sub feature vector of dimension 250 with binary values indicting what top frequent words in that category occur in the review. In other words, commonWords in hw3 uses a vector [commonWords[0] in review, commonWords[1] in review, ..., commonWords[499] in review]. I simply add [topFrequentWords_bycategory[0] in review, topFrequentWords_bycategory [1] in review, ..., topFrequentWords_bycategory [249] in review]. From first part which implements commonWords, I get a vector of dimension 500. Then I enter the for loop for the 10 categories, and each category has a vector of dimension 250. I append the sub feature vector to the 500 commonWords vector after each iteration. After the 10 iterations for each category, I now get a vector with dimension of 500+2500=3000. This whole feature vector of dimension 3000 is my final feature vector. I then use this feature function to create x_train and x_validation for my multiclass svm. The code is as follow:

CSE 158 Assignment 1 Report
Qihan Guan A92413483
Kaggle Leaderboard Name: Qihan Guan
Kaggle profile name: Qihan Guan (qiguan)

```
x_train = [feature(r) for r in review_train]
y_train = [r['categoryID'] for r in review_train]

x_validation = [feature(r) for r in review_validation]
y_validation = [r['categoryID'] for r in review_validation]
clf = OneVsRestClassifier(LinearSVC(C=0.1,random_state=0))
clf.fit(x_train, y_train)
predict = clf.predict(x_validation)
acc = [(x==y) for (x,y) in zip(predict, y_validation)]
acc = sum(acc)*1.0/len(acc)

print(acc)
```

During the validation process, I tune my 3 parameters. First one is how many common words I need overall. Second one is how many top frequent words per category I need. Third one is the parameter c for multiclass svm. In the end, when I choose 500 common words, 250 most frequent words per category, and c=0.1, I have the best validation accuracy. This also gives me the best Kaggle score of 0.64159.

Side notes:
For the feature vector design, I originally add some user activity levels by category. This feature vector gives me very high validation accuracy, which is as high as 0.67. However, when I submit it to Kaggle, the test score returned is a merely 0.49. I believe the reason of this is that in the test data, there are many users we might have never seen before. When adding user activity features, my model heavily correlates to training and validation data, but does not generalize well on new data. Therefore, I remove the user features, and only improve word features. User features can be useful when designing it right, however, my design of user features have some flaw so I removed it.