CSE258_HW4

March 6, 2017

1 CSE258 HW 4

```
In [1]: import numpy as np
        import urllib
        import string
        import nltk
        from nltk import bigrams
        from collections import defaultdict
        from sklearn import linear_model
        from sklearn.metrics import mean_squared_error
        from math import log10
        from scipy import spatial
```

```
In [2]: def parseData(fname):
            for l in urllib.urlopen(fname):
                yield eval(l)

        print "Reading data..."
        data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_5000
        print "done"
```

```
Reading data...
done
```

1.0.1 Task 1

There are 182246 unique bigrams among all of the reviews.

The 5 most-frequently-occurring bigrams along with their number of occurrences:
The bigram (‘with’, ‘a’) has an occurence of 4587
The bigram (‘in’, ‘the’) has an occurence of 2595
The bigram (‘of’, ‘the’) has an occurence of 2245
The bigram (‘is’, ‘a’) has an occurence of 2056
The bigram (‘on’, ‘the’) has an occurence of 2033

```
In [3]: # bigram count
        punctuation = set(string.punctuation)
        bigramCount = defaultdict(int)
```

```
for d in data:
    text = ''.join([c for c in d['review/text'].lower() if not c in punctua
    bg = list(bigrams(text.split())) # all bigrams in text
    for b in bg:
        bigramCount[b] += 1
print 'There are ' + str(len(bigramCount)) + ' unique bigrams among all of
# sort
bigramSorted = list(sorted(bigramCount, key = lambda x : bigramCount[x], re
for i in range(0,5):
    print 'The bigram ' + str(bigramSorted[i]) + ' has an occurence of ' +
```

```
There are 182246 unique bigrams among all of the reviews
The bigram ('with', 'a') has an occurence of 4587
The bigram ('in', 'the') has an occurence of 2595
The bigram ('of', 'the') has an occurence of 2245
The bigram ('is', 'a') has an occurence of 2056
The bigram ('on', 'the') has an occurence of 2033
```

1.0.2 Task 2

The MSE obtained using bigrams only is : 0.34361068509441478

```
In [4]: Bigrams = [x for x in bigramSorted[:1000]]
        bigramID = dict(zip(Bigrams, range(len(Bigrams))))
        bigramSet = set(Bigrams)

        def feat(data):
            feat = [0] * len(Bigrams)
            text = ''.join([c for c in data['review/text'].lower() if not c in punc
            bg = list(bigrams(text.split())) # all bigrams in text
            for b in bg:
                if b in bigramSet:
                    feat[bigramID[b]] += 1
            feat.append(1) #offset
            return feat
```

```
In [5]: x = [feat(d) for d in data]
        y = [d['review/overall'] for d in data]

        clf = linear_model.Ridge(1.0, fit_intercept=False)
        clf.fit(x, y)
        theta = clf.coef_
        predictions = clf.predict(x)

        mean_squared_error(predictions, y)
```

```
Out[5]: 0.34361068509441478
```

1.0.3 Task 3

The MSE obtained using unigrams and bigrams is : 0.28933386918744819

```
In [6]: # unigram count
        wordCount = defaultdict(int)

        for d in data:
            text = ''.join([c for c in d['review/text'].lower() if not c in punctua
            for w in text.split():
                wordCount[w] += 1
```

```
In [7]: # merge unigram and bigram
        mergeCount = dict(wordCount)
        mergeCount.update(bigramCount)

        mergeSorted = list(sorted(mergeCount, key = lambda x : mergeCount[x], rever

        grams = [x for x in mergeSorted[:1000]]
        gramID = dict(zip(grams, range(len(grams))))
        gramSet = set(grams)

        def feat(data):
            feat = [0] * len(grams)
            text = ''.join([c for c in data['review/text'].lower() if not c in punc
            bg = list(bigrams(text.split())) # all bigrams in text
            for w in text.split() + bg:
                if w in gramSet:
                    feat[gramID[w]] += 1
            feat.append(1) #offset
            return feat
```

```
In [8]: x = [feat(d) for d in data]
        y = [d['review/overall'] for d in data]

        clf = linear_model.Ridge(1.0, fit_intercept=False)
        clf.fit(x, y)
        theta = clf.coef_
        predictions = clf.predict(x)

        mean_squared_error(predictions, y)
```

```
Out[8]: 0.28933386918744819
```

1.0.4 Task 4

The 5 unigrams/bigrams with the most positive associated weights are :
‘sort’ has associated weight of 0.521680776822
(‘a’, ‘bad’) has associated weight of 0.226288834348

('of', 'these') has associated weight of 0.22289001188
('not', 'bad') has associated weight of 0.216268615711
('the', 'best') has associated weight of 0.213772219036
The 5 unigrams/bigrams with the most negative associated weights are :
('sort', 'of') has associated weight of -0.645937945334
'water' has associated weight of -0.271900176951
'corn' has associated weight of -0.23756003904
('the', 'background') has associated weight of -0.218138672448
'straw' has associated weight of -0.199753548917

```
In [9]: weights = zip(theta[0:-1], range(1000))
        weights.sort()

        print 'The 5 unigrams/bigrams with the most positive associated weights are
        for i in range(0,5):
            print  str(grams[weights[-i-1][1]]) + ' has associated weight of ' + st
        print '\n The 5 unigrams/bigrams with the most negative associated weights
        for i in range(0,5):
            print  str(grams[weights[i][1]]) + ' has associated weight of ' + str(w
```

```
The 5 unigrams/bigrams with the most positive associated weights are :
sort has associated weight of 0.521680776822
('a', 'bad') has associated weight of 0.226288834348
('of', 'these') has associated weight of 0.22289001188
('not', 'bad') has associated weight of 0.216268615711
('the', 'best') has associated weight of 0.213772219036

 The 5 unigrams/bigrams with the most negative associated weights are :
('sort', 'of') has associated weight of -0.645937945334
water has associated weight of -0.271900176951
corn has associated weight of -0.23756003904
('the', 'background') has associated weight of -0.218138672448
straw has associated weight of -0.199753548917
```

1.0.5 Task 5

The inverse document frequency of the words 'foam', 'smell', 'banana', 'lactic', and 'tart' are :
[1.1378686206869628, 0.5379016188648442, 1.6777807052660807, 2.9208187539523753, 1.8068754016455384]
Their tf-idf scores in the first review (using log base 10) are:
[2.2757372413739256, 0.5379016188648442, 3.3555614105321614, 5.841637507904751, 1.8068754016455384]

```
In [10]: wordList = ['foam', 'smell', 'banana', 'lactic', 'tart']
         N = len(data)
         # inverse document frequency
         def idf(w):
```

```
        count = 0
        for d in data:
            text = ''.join([c for c in d['review/text'].lower() if not c in pu
            if w in text.split():
                count += 1
        return log10(N * 1. / count)

    idfList = [idf(w) for w in wordList]
    print idfList
```

```
[1.1378686206869628, 0.5379016188648442, 1.6777807052660807, 2.9208187539523753, 1.
```

```
In [11]: # term frequency in first review
         def tf(w):
             count = 0
             text = ''.join([c for c in data[0]['review/text'].lower() if not c in
             for t in text.split():
                 if t == w:
                     count += 1
             return count

         tfList = [tf(w) for w in wordList]
```

```
In [12]: [i * j for i,j in zip(idfList, tfList)]
```

```
Out[12]: [2.2757372413739256,
          0.5379016188648442,
          3.3555614105321614,
          5.841637507904751,
          1.8068754016455384]
```

1.0.6 Task 6

The cosine similarity between the first and the second review in terms of their tf-idf representations is : 0.10613024167865803

```
In [13]: counts = [(wordCount[w], w) for w in wordCount]
         counts.sort()
         counts.reverse()

         # tfidf feature words -- 1000 most common
         featWords = [x[1] for x in counts[:1000]]
         featID = dict(zip(featWords, range(1000)))

         # idf for feature words
         countList = [0] * 1000
         for d in data:
             text = ''.join([c for c in d['review/text'].lower() if not c in punctu
```

```
        textWord = set(text.split())
        for t in textWord:
            if t in featID:
                countList[featID[t]] += 1
    idfList = [log10(N * 1. / x) for x in countList]
```

```
In [14]: # tfide feature extraction
         def tfidfFeat(d):
             count = [0] * len(featWords) # tf counts
             text = ''.join([c for c in d['review/text'].lower() if not c in punctu
             for t in text.split():
                 if t in featWords:
                     count[featID[t]] += 1
             feat = [i * j for i,j in zip(idfList, count)]
         #      feat.append(1)
             return feat
```

```
In [15]: feat1 = tfidfFeat(data[0])
         feat2 = tfidfFeat(data[1])
         1 - spatial.distance.cosine(feat1, feat2)
```

```
Out[15]: 0.10613024167865803
```

1.0.7 Task 7

beerID : 52211
profileName : Dope
reviewText : A: A hazy deep orange pour, almost red. Small white head that fades quickly. A little spotty lacing. S: Big pumpkin, cinnamon, ginger, nutmeg and brown sugar. Sweet. Smells like a pumpkin pie mixed with a gingerbread cookie. T: Tons of pumpkin dominates throughout. Cinnamon, ginger, nutmeg and a bit of vanilla creaminess. M: Smooth medium body. Tiny bit of drying alcohol. O: Excellent pumpkin ale. Heavy on the pumpkin but the spices don't get completely overshadowed either.

```
In [17]: cosineList = [1 - spatial.distance.cosine(feat1, tfidfFeat(x)) for x in da
         result = zip(cosineList, range(len(cosineList)))
         result.sort()
         result.reverse()
         print 'beerID : ' + str(data[result[0][1]]['beer/beerId']) + '\n' + 'profi
         print 'reviewText : ' + data[result[0][1]]['review/text']
```

```
beerID : 52211
profileName : Dope

reviewText : A: A hazy deep orange pour, almost red. Small white head that fades qu
```

1.0.8 Task 8

The MSE obtained with the 1000-dimensional tf-idf representations is : 0.27875956007772285

```
In [18]: # rewrite the tfide feature extraction with offset
         def tfidfFeat(d):
             count = [0] * len(featWords) # tf counts
             text = ''.join([c for c in d['review/text'].lower() if not c in punctu
             for t in text.split():
                 if t in featWords:
                     count[featID[t]] += 1
             feat = [i * j for i,j in zip(idfList, count)]
             feat.append(1)
             return feat
```

```
In [19]: x = [tfidfFeat(d) for d in data]
         y = [d['review/overall'] for d in data]

         clf = linear_model.Ridge(1.0, fit_intercept=False)
         clf.fit(x, y)
         theta = clf.coef_
         predictions = clf.predict(x)

         mean_squared_error(predictions, y)
```

```
Out[19]: 0.27875956007772285
```

```
In [ ]:
```