HW4

November 20, 2018

In [1]:
```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
import ast
import math
from nltk.stem.porter import *
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

0.0.1 Question 1

In [2]:
```
def parseData(fname):
  for l in open(fname):
    yield ast.literal_eval(l)
```

In [3]:
```
### Just the first 5000 reviews

print("Reading data...")
data = list(parseData("beer_50000.json"))
print("done")

# There's one review without text. Removing it
first5000 = data[:5001]
first5000.remove(first5000[3499])
```

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

In [171]:
```
### Ignore capitalization and remove punctuation

  bigramCount = defaultdict(int)
  punctuation = set(string.punctuation)
```

```
        for d in first5000:
            r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
            words = r.split()
            for i in range(0, len(words)-1):
                bigramCount[words[i] + ' ' + words[i+1]] += 1
```

In [172]:
```
mostCommonBigrams = [(v, bigramCount[v]) for v in bigramCount]
mostCommonBigrams = sorted(mostCommonBigrams, key=lambda x:-x[1])
```

In [173]:
```
mostCommonBigrams[:5]
```

Out[173]:
```
[('with a', 4588),
 ('in the', 2597),
 ('of the', 2245),
 ('is a', 2057),
 ('on the', 2034)]
```

0.0.2 Question 2

In [174]:
```
bigrams = [x[0] for x in mostCommonBigrams[:1000]]
```

In [175]:
```
### Sentiment analysis

bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)

def feature(datum):
    feat = [0]*len(bigrams)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
    words = r.split()
    for i in range(0, len(words)-1):
        b = words[i] + ' ' + words[1]
        if b in bigrams:
            feat[bigramId[b]] += 1
    feat.append(1) #offset
    return feat

X = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
```

In [176]:
```
#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

In [177]:
```
print(mean_squared_error(y, predictions))
```

```
0.472180192210667
```

0.0.3 Question 3

```
In [17]: def tf(word, sentence):
             return len([x for x in sentence.split() if x == word])/len(sentence.split())

In [18]: def idf(word, corpus):
             value = 0
             for s in corpus:
                 if word in s:
                     value += 1
             if value == 0:
                 return -1
             else:
                 return math.log(len(corpus)/value, 10)

In [227]: corpus = [d['review/text'] for d in first5000]
          for i in range(0,len(corpus)):
              r = ''.join([c for c in corpus[i].lower() if not c in punctuation])
              corpus[i] = ''
              for word in r.split():
                  corpus[i] += ' ' + word

In [181]: for word in ['foam', 'smell', 'banana', 'lactic', 'tart']:
              wordIDF = idf(word, corpus)
              wordTF = tf(word, corpus[0])
              print(word)
              print('idf = ' + str(wordIDF))
              print('tfidf = ' + str(wordTF*wordIDF))
              print()
```

```
foam
idf = 0.9476909003526764
tfidf = 0.03868126123888475

smell
idf = 0.38090666937325723
tfidf = 0.007773605497413412

banana
idf = 1.5751183633689327
tfidf = 0.0642905454436299

lactic
idf = 2.920818753952375
tfidf = 0.11921709199805611

tart
idf = 1.0078885122130503
tfidf = 0.020569153310470413
```

0.0.4 Question 4

```
In [182]: allWords = set(corpus[0].split() + corpus[1].split())

In [183]: review1 = []
          review2 = []
          for word in allWords:
              wordIDF = idf(word, corpus)
              review1TF = tf(word, corpus[0])
              review2TF = tf(word, corpus[1])
              review1.append(review1TF*wordIDF)
              review2.append(review2TF*wordIDF)

In [184]: dot = []
          r1magnitude = []
          r2magnitude = []
          for i in range(0, len(review1)):
              dot.append(review1[i]*review2[i])
              r1magnitude.append(review1[i]*review1[i])
              r2magnitude.append(review2[i]*review2[i])
          dot = sum(dot)
          r1magnitude = math.sqrt(sum(r1magnitude))
          r2magnitude = math.sqrt(sum(r2magnitude))

In [185]: cosine = dot/(r1magnitude*r2magnitude)

In [186]: cosine

Out[186]: 0.05716312389736702
```

0.0.5 Question 5

```
In [187]: def cosSimilarity(v1, v2):
              dot = []
              v1magnitude = []
              v2magnitude = []
              for i in range(0, len(v1)):
                  dot.append(v1[i]*v2[i])
                  v1magnitude.append(v1[i]*v1[i])
                  v2magnitude.append(v2[i]*v2[i])
              dot = sum(dot)
              v1magnitude = math.sqrt(sum(v1magnitude))
              v2magnitude = math.sqrt(sum(v2magnitude))
              return dot/(v1magnitude*v2magnitude)
```

```
In [188]: cosSimilarities = []
          for i in range(1, len(corpus)):
              review1 = []
              review2 = []
              allWords = set(corpus[0].split() + corpus[i].split())
              for word in allWords:
                  wordIDF = idf(word, corpus)
                  review1TF = tf(word, corpus[0])
                  review2TF = tf(word, corpus[i])
                  review1.append(review1TF*wordIDF)
                  review2.append(review2TF*wordIDF)
              cosSimilarities.append(cosSimilarity(review1, review2))

In [189]: cosSimilarities.index(max(cosSimilarities))

Out[189]: 2342

In [190]: print('profileName: ' + data[2342+1]['user/profileName'])
          print('beerId: ' + data[2342+1]['beer/beerId'])

profileName: spicelab
beerId: 72146
```

0.0.6 Question 6

```
In [191]: ### Just take the most popular words...

          wordCount = defaultdict(int)
          punctuation = set(string.punctuation)
          for d in first5000:
            r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
            for w in r.split():
              wordCount[w] += 1

          counts = [(wordCount[w], w) for w in wordCount]
          counts.sort()
          counts.reverse()

          words = [x[1] for x in counts[:1000]]

In [192]: ### Sentiment analysis

          wordId = dict(zip(words, range(len(words))))
          wordSet = set(words)

          wordsIDFs = []
          for w in words:
              wordsIDFs.append(idf(w, corpus))
```

```
In [225]: def feature(datum):
              feat = []
              for i in range(0, len(words)):
                  wordIDF = wordsIDFs[i]
                  wordTF = tf(words[i], corpus[first5000.index(datum)])
                  feat.append(wordTF*wordIDF)
              feat.append(1) #offset
              return feat

In [228]: X = [feature(d) for d in first5000]
          y = [d['review/overall'] for d in first5000]

In [229]: #With regularization
          clf = linear_model.Ridge(1.0, fit_intercept=False)
          clf.fit(X, y)
          theta = clf.coef_
          predictions = clf.predict(X)

In [230]: print(mean_squared_error(y, predictions))

0.5208065538083951
```

0.0.7 Question 7

```
In [4]: dataCopy = data
        train = []
        validation = []
        test = []
        for i in range(0, 5000):
            index = random.randint(0, len(dataCopy))
            train.append(dataCopy[i])
            dataCopy.remove(dataCopy[i])
            index = random.randint(0, len(dataCopy))
            validation.append(dataCopy[i])
            dataCopy.remove(dataCopy[i])
            index = random.randint(0, len(dataCopy))
            test.append(dataCopy[i])
            dataCopy.remove(dataCopy[i])

In [5]: ### Ignore capitalization and remove punctuation

        bigramCount = defaultdict(int)
        punctuation = set(string.punctuation)
        for d in train:
            r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
            words = r.split()
            for i in range(0, len(words)-1):
                bigramCount[words[i] + ' ' + words[i+1]] += 1
```

```
In [6]: mostCommonBigrams = [(v, bigramCount[v]) for v in bigramCount]
        mostCommonBigrams = sorted(mostCommonBigrams, key=lambda x:-x[1])

In [7]: bigramsNoPunct = [v[0] for v in mostCommonBigrams[:1000]]

In [8]: ### Ignore capitalization and preserve punctuation
        import re

        def strip(s):
            return s.strip()

        bigramCount = defaultdict(int)
        punctuation = set(string.punctuation)
        for d in train:
            r = ''.join([c for c in d['review/text'].lower()])
            words = [item for item in map(strip, re.split("(\W+)", r)) if len(item) > 0]
            for i in range(0, len(words)-1):
                bigramCount[words[i] + ' ' + words[i+1]] += 1

In [9]: mostCommonBigrams = [(v, bigramCount[v]) for v in bigramCount]
        mostCommonBigrams = sorted(mostCommonBigrams, key=lambda x:-x[1])

In [10]: bigramsPunct = [v[0] for v in mostCommonBigrams[:1000]]

In [11]: bigrams = [bigramsNoPunct, bigramsPunct]

In [12]: ### Just take the most popular words without punctuation

         wordCount = defaultdict(int)
         punctuation = set(string.punctuation)
         for d in train:
           r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
           for w in r.split():
             wordCount[w] += 1

         counts = [(wordCount[w], w) for w in wordCount]
         counts.sort()
         counts.reverse()

         wordsNoPunctuation = [x[1] for x in counts[:1000]]

In [13]: ### Just take the most popular words presereving punctiation

         wordCount = defaultdict(int)
         punctuation = set(string.punctuation)
         for d in train:
             r = ''.join([c for c in d['review/text'].lower()])
             words = [item for item in map(strip, re.split("(\W+)", r)) if len(item) > 0]
             for w in r.split():
```

```
            wordCount[w] += 1

        counts = [(wordCount[w], w) for w in wordCount]
        counts.sort()
        counts.reverse()

        wordsPunctuation = [x[1] for x in counts[:1000]]
```

In [14]:
```
unigrams = [wordsNoPunctuation, wordsPunctuation]
```

In [15]:
```
grams = [unigrams, bigrams]
```

In [20]:
```
corpusNoPunct = []
corpusPunct = []
for i in range(0, len(train)):
    text = train[i]['review/text']
    r = ''.join([c for c in text.lower() if not c in punctuation])
    corpusNoPunct.append('')
    for word in r.split():
        corpusNoPunct[i] += ' ' + word

    corpusPunct.append('')
    for word in ''.join([c for c in text.lower()]).split():
        corpusPunct[i] += ' ' + word
```

In [21]:
```
corpus = [corpusNoPunct, corpusPunct]
```

In [22]:
```
### Getting IDFs

IDFsTable = []
# unigram no punct
# unigram punct
# bigram no punct
# bigram punct

gramsIds = []

punct = 0
for g in grams:
    for j in g:
        words = j
        wordId = dict(zip(words, range(len(words))))
        gramsIds.append(wordId)

        wordsIDFs = []
        for w in words:
            wordsIDFs.append(idf(w, corpus[punct]))
        IDFsTable.append(wordsIDFs)
        punct = (punct + 1)%2
```

```
In [23]: def featureUnigram(datum, punct):
             feat = [0]*len(grams[0][punct])
             if punct == 0:
                 r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
             else:
                 r = ''.join([c for c in datum['review/text'].lower()])
             for w in r.split():
                 if w in grams[0][punct]:
                     feat[gramsIds[punct][w]] += 1
             feat.append(1) #offset
             return feat

In [24]: def featureBigrams(datum, punct):
             feat = [0]*len(grams[1][punct])
             if punct == 0:
                 r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
             else:
                 r = ''.join([c for c in datum['review/text'].lower()])
             words = r.split()
             for i in range(0, len(words)-1):
                 b = words[i] + ' ' + words[1]
                 if b in bigrams:
                     feat[gramsIds[2+punct][b]] += 1
             feat.append(1) #offset
             return feat

In [25]: def featureTFIDF(i, gram, punct):
             #gram = 0 -> unigram
             #gram = 1 -> bigram
             sentence = corpus[punct][i]
             if sentence == '':
                 return [0]*len(grams[gram][punct]) + [1]

             feat = []
             for gramIndex in range(0, len(grams[gram][punct])):
                 currentGram = grams[gram][punct][gramIndex]
                 gramIDF = IDFsTable[2*gram+punct][gramsIds[2*gram+punct][currentGram]]
                 gramTF = tf(currentGram, sentence)
                 feat.append(gramTF*gramIDF)
             feat.append(1) #offset
             return feat

In [26]: # running models
         thetas = []
         mses = []
         allModels = []
         for x in [0,1]: # unigram x bigram
             for y in [0,1]: # noPunct x Punct
```

```
                for z in [0,1]: # tfidf x counts
                    if z == 0:
                        X_train = [featureTFIDF(corpusIndex, x, y) for corpusIndex in range(0,l
                        X_validation = [featureTFIDF(corpusIndex, x, y) for corpusIndex in rang
                        X_test = [featureTFIDF(corpusIndex, x, y) for corpusIndex in range(0,le
                    else:
                        if x == 0:
                            X_train = [featureUnigram(d, y) for d in train]
                            X_validation = [featureUnigram(d, y) for d in validation]
                            X_test = [featureUnigram(d, y) for d in test]
                        else:
                            X_train = [featureBigrams(d, y) for d in train]
                            X_validation = [featureBigrams(d, y) for d in validation]
                            X_test = [featureBigrams(d, y) for d in test]

                    y_train = [d['review/overall'] for d in train]
                    y_validation = [d['review/overall'] for d in validation]
                    y_test = [d['review/overall'] for d in test]

                    models = []
                    for c in [0.01, 0.1, 1, 10, 100]:
                        clf = linear_model.Ridge(c, fit_intercept=False)
                        clf.fit(X_train, y_train)
                        theta = clf.coef_
                        predictions = clf.predict(X_validation)
                        mse = mean_squared_error(y_validation, predictions)
                        models.append([clf, mse])
                    allModels.append(models)
                    models = sorted(models, key=lambda x:x[1])
                    bestModel = models[0][0]
                    thetas.append(bestModel.coef_)
                    testPreds = bestModel.predict(X_train)
                    mse = mean_squared_error(y_train, testPreds)
                    mses.append(mse)
```

```
In [29]: mses
```

```
Out[29]: [0.3827984754961387,
          0.33818140493974114,
          0.41449847824164754,
          0.3567822877286595,
          0.5791860002151384,
          0.5791860002151384,
          0.5791860002151384,
          0.5791860002151384]
```

Unigrams	tfidf	word counts
No Punctuation	0.382798	0.338181
Punctuation	0.414498	0.356782

Bigrams	tfidf	word counts
No Punctuation	0.579186	0.579186
Punctuation	0.579186	0.579186