CSE258 Homework 4

Chenyu Huang, University of California San Diego 03/05/2017

1 Homework 4

Task 1 The 5 most common bigrams in the first 5000 reviews are:

```
[('with a', 4587), ('in the', 2595), ('of the', 2245), ('is a', 2056), ('on the', 2033)]
```

Figure 1: 5 most common bigrams

Task 2 Using the most 1000 bigrams from the corpus, the MSE of the linear model is 0.343313.

Task 3 With a mixture of the most common bigrams and unigrams, the MSE of the model is 0.289394.

Task 4 The 5 words with the most common positive weights and 5 words with most negative weights are as follows:

Top 5 words with the most positive weights:

$$('sort', 0.50408647668155093), \tag{1}$$
$$('abad', 0.22799617008799439), \tag{2}$$
$$('ofthese', 0.22066627419916279), \tag{3}$$
$$('notbad', 0.21355987652123357), \tag{4}$$
$$('thebest', 0.20916785548604572) \tag{5}$$

Top 5 words with the most negative weights:

$$('straw', -0.19930067914769131), \tag{6}$$
$$('thebackground', -0.22131852632951593), \tag{7}$$
$$('corn', -0.23815182761392634), \tag{8}$$
$$('water', -0.27289904891323319), \tag{9}$$
$$('sortof', -0.63021419493511976) \tag{10}$$

Task 5 The IDF of the words are:

$$('foam' : 1.1139433523068367), \tag{11}$$
$$('smell' : 0.47712125471966244), \tag{12}$$
$$('banana' : 1.6720978579357175), \tag{13}$$
$$('lactic' : 2.9206450014067875), \tag{14}$$
$$('tart' : 1.806179973983887) \tag{15}$$

The TF-IDF of the words in the first review are:

$$('foam' : 2.2278867046136734), \tag{16}$$
$$('smell' : 0.47712125471966244), \tag{17}$$
$$('banana' : 3.344195715871435), \tag{18}$$
$$('lactic' : 5.841290002813575), \tag{19}$$
$$('tart' : 1.806179973983887) \tag{20}$$

Task 6 Using just the 1000 most common unigram feature representation. The cosine similarity between the first and second reviews is: 0.106130241679.

Task 7 Using only the 1000 most common unigram feature representation. The review with the highest cosine similarity with the first review has the following attributes:

$$BeerId : 52211 \tag{21}$$
$$Profile\ Name : Heatwave33 \tag{22}$$

Task 8 Using tf-idf as their feature representation, the MSE of the unigram model now is: 0.278742

2 Code

Listing 1: task1

```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import operator

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

bigramCount = {}
### Count bigrams
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  r = r.split()
  for i in range(0, len(r) - 1):
    pair = r[i] + " " + r[i+1]
    if pair in bigramCount:
      bigramCount[pair] = bigramCount[pair] + 1
    else:
      bigramCount[pair] = 1

print len(bigramCount)
sorted_bigram = sorted(bigramCount.items(), key=operator.itemgetter(1),
    reverse = True)
print sorted_bigram[:5]
```

Listing 2: task2

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import operator

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

bigramCount = {}
### Count bigrams
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  r = r.split()
  for i in range(0, len(r) - 1):
    pair = r[i] + " " + r[i+1]
    if pair in bigramCount:
      bigramCount[pair] = bigramCount[pair] + 1
    else:
      bigramCount[pair] = 1

sorted_bigram = sorted(bigramCount.items(), key = operator.itemgetter(1),
    reverse = True)

words = []
for s in sorted_bigram[:1000]:
  words.extend([s[0]])

### Sentiment analysis

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


def feature(datum):
  feat = [0] * len(words)
```

```
  r = ''.join([c for c in datum['review/text'].lower() if not c in
      punctuation])
  r = r.split()
  for i in range(0, len(r) - 1):
    w = r[i] + " " + r[i+1]
    if w in words:
      feat[wordId[w]] += 1
  feat.append(1) #offset
  return feat

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

#No regularization
#theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
# predictions = clf.predict(X)
# The mean squared error
print("Mean squared error: %f"
      % np.mean((clf.predict(X) - y) ** 2))
```

Listing 3: task3 and 4

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import operator

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

bigramCount = {}
### Count bigrams
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  r = r.split()
  for i in range(0, len(r) - 1):
    pair = r[i] + " " + r[i+1]
    if pair in bigramCount:
      bigramCount[pair] = bigramCount[pair] + 1
    else:
      bigramCount[pair] = 1

unigramCount = {}
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in r.split():
    if w in unigramCount:
      unigramCount[w] += 1
    else:
      unigramCount[w] = 1


combine = dict(bigramCount.items() + unigramCount.items())
combine_sort = sorted(combine.items(), key = operator.itemgetter(1), reverse
    = True)

words = []
```

```
for s in combine_sort[:1000]:
  words.extend([s[0]])

### Sentiment analysis

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

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

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

#No regularization
#theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
print("Mean squared error: %f"
      % np.mean((clf.predict(X) - y) ** 2))

weights = {}
for i in range(len(words)):
  for key, value in wordId.iteritems():
    if value == i:
      weights[key] = theta[i]
      break

weight_sort = sorted(weights.items(), key = operator.itemgetter(1), reverse =
     True)
print weight_sort[:5]
print weight_sort[-5:]
```

Listing 4: task5

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import math

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

idf = {}
for word in ['foam', 'smell', 'banana', 'lactic', 'tart']:
  idf[word] = 0

punctuation = set(string.punctuation)
for word in ['foam', 'smell', 'banana', 'lactic', 'tart']:
  for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation
        ])
    r = r.split()
    if word in r:
      idf[word] += 1

for key in idf:
  idf[key] = math.log10(len(data) / idf[key])
print idf

tf = {'foam':0, 'smell':0, 'banana':0, 'lactic':0, 'tart':0}
tf_idf = {}

for key in idf:
  d = data[0]
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  r = r.split()
  for w in r:
    if key == w:
      tf[key] += 1
  tf_idf[key] = tf[key] * idf[key]
```

```
print tf_idf
```

Listing 5: task6

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import math

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

### 1000 most common unigrams
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
  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 = set([x[1] for x in counts[:1000]])
idf = {}
tf1 = {}
for word in ''.join([c for c in data[0]['review/text'].lower() if not c in
    punctuation]).split():
  # only using 1000 features
  if word in words:
    if word in tf1:
      tf1[word] += 1
    else:
      tf1[word] = 1
    idf[word] = 0

tf2 = {}
for word in ''.join([c for c in data[1]['review/text'].lower() if not c in
    punctuation]).split():
```

```
  # only using 1000 features
  if word in words:
    if word in tf2:
      tf2[word] += 1
    else:
      tf2[word] = 1
    idf[word] = 0

for word in idf:
  for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation
        ])
    r = r.split()
    if word in r:
      idf[word] += 1

for key in idf:
  idf[key] = math.log10(5000 / float(idf[key]))

tf_idf1 = {}
for key in tf1:
  tf_idf1[key] = tf1[key] * idf[key]

tf_idf2 = {}
for key in tf2:
  tf_idf2[key] = tf2[key] * idf[key]

def find_cosine_sim(tf_idf1, tf_idf2):
  nominator = 0.0
  deno_part1 = 0.0
  deno_part2 = 0.0
  for key in tf_idf1:
    if key in tf_idf2:
      nominator += tf_idf1[key] * tf_idf2[key]
  for key in tf_idf1:
      deno_part1 += tf_idf1[key] ** 2
  for key in tf_idf2:
      deno_part2 += tf_idf2[key] ** 2
  denominator = math.sqrt(deno_part1) * math.sqrt(deno_part2)
  return nominator / denominator

print find_cosine_sim(tf_idf1, tf_idf2)
```

Listing 6: task7

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import math

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

### 1000 most common unigrams
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
  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 = set([x[1] for x in counts[:1000]])

idf = {}
for w in words:
  idf[w] = 0

tf1 = {}
for word in ''.join([c for c in data[0]['review/text'].lower() if not c in
    punctuation]).split():
  # only using 1000 features
  if word in words:
    if word in tf1:
      tf1[word] += 1
    else:
      tf1[word] = 1
```

```
for word in idf:
  for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation
        ])
    r = r.split()
    if word in r:
      idf[word] += 1


for key in idf:
  idf[key] = math.log10(5000 / float(idf[key]))

tf_idf1 = {}
for key in tf1:
  tf_idf1[key] = tf1[key] * idf[key]

def find_cosine_sim(tf_idf1, tf_idf2):
  nominator = 0.0
  deno_part1 = 0.0
  deno_part2 = 0.0
  for key in tf_idf1:
    if key in tf_idf2:
      nominator += tf_idf1[key] * tf_idf2[key]
  for key in tf_idf1:
      deno_part1 += tf_idf1[key] ** 2
  for key in tf_idf2:
      deno_part2 += tf_idf2[key] ** 2
  denominator = math.sqrt(deno_part1) * math.sqrt(deno_part2)
  if denominator == 0:
    return 0
  return nominator / denominator

counter = 1
sim = 0
text = ""
bid = ""
pname = ""
index = 1

for d in data[1:]:
  tf2 = {}
  tf_idf2 = {}
  for word in ''.join([c for c in d['review/text'].lower() if not c in
      punctuation]).split():
    if word in words:
      if word in tf2:
        tf2[word] += 1
      else:
        tf2[word] = 1
  for key in tf2:
    tf_idf2[key] = tf2[key] * idf[key]
```

```
  if find_cosine_sim(tf_idf1, tf_idf2) > sim:
    sim = find_cosine_sim(tf_idf1, tf_idf2)
    text = d['review/text']
    index = counter
    bid = d['beer/beerId']
    pname = d['user/profileName']
  counter += 1

print text
print bid
print pname
```

Listing 7: task8

```
import numpy as np
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import operator

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

### Just the first 5000 reviews
print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"

### 1000 most common unigrams
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
  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 = set([x[1] for x in counts[:1000]])

idf = {}
for w in words:
  idf[w] = 0

for word in idf:
  for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation
        ])
    r = r.split()
    if word in r:
      idf[word] += 1

## Sentiment analysis
wordId = dict(zip(words, range(len(words))))
def feature(datum):
```

```
  feat = [0] * len(words)
  tf = {}
  for word in ''.join([c for c in datum['review/text'].lower() if not c in
      punctuation]).split():
  # only using 1000 features
    if word in words:
      if word in tf:
        tf[word] += 1
      else:
        tf[word] = 1
  tf_idf = {}
  for key in tf:
    tf_idf[key] = tf[key] * idf[key]
    feat[wordId[key]] = tf_idf[key]
  feat.append(1) #offset
  return feat

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

#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
# predictions = clf.predict(X)
# The mean squared error
print("Mean squared error: %f"
      % np.mean((clf.predict(X) - y) ** 2))
```