hw4

November 25, 2019

[1]:
```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict # Dictionaries with default values
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import ast
```

[2]:
```
def parseData(fname):
  for l in open(fname):
    yield eval(l)

def parseDataFromURL(fname):
  for l in urlopen(fname):
    yield eval(l)
```

[3]:
```
data = list(parseData("train_Category.json"))[:10000]
data[1]
```

[3]:
```
{'n_votes': 0,
 'review_id': 'r24440074',
 'user_id': 'u08070901',
 'review_text': 'Pretty decent. The ending seemed a little rush but a good
ending to the first trilogy in this series. The fact that most of the time it is
a military fantasy makes it interesting. Also all of the descriptions of food
just make me hungry.',
 'rating': 5,
 'genreID': 2,
 'genre': 'fantasy_paranormal'}
```

[4]:
```
### Ignore capitalization and remove punctuation

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
print(len(wordCount))

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

```
73286
```

1. How many unique bigrams are there amongst the reviews? List the 5 most-frequently-occurring bigrams along with their number of occurrences in the corpus (1 mark).

[5]:
```
import nltk
from nltk.util import ngrams
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures


bgc = defaultdict(int)
punct = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review_text'].lower() if not c in punct])
    bigrm = nltk.bigrams(r.split())
    for w in bigrm:
        bgc[w] +=1
```

[6]:
```
bcounts = [(bgc[w], w) for w in bgc]
bcounts.sort()
bcounts.reverse()
bcounts[:5]
```

[6]:
```
[(7927, ('of', 'the')),
 (5850, ('this', 'book')),
 (5627, ('in', 'the')),
 (3189, ('and', 'the')),
 (3183, ('is', 'a'))]
```

2. The code provided performs least squares using the 1000 most common unigrams. Adapt it to use the 1000 most common bigrams and report the MSE obtained using the new predictor (use bigrams only, i.e., not unigrams+bigrams) (1 mark). Note that the code performs regularized regression with a regularization parameter of 1.0. The prediction target should be the 'rating' field in each review.

[7]:
```
words = [x[1] for x in bcounts[:1000]]
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

[ ]:
```

```

[8]:
```
reviews=[]
punct = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review_text'].lower() if not c in punctuation])
    reviews.append(r)
reviews[1]
```

[8]:
```
'pretty decent the ending seemed a little rush but a good ending to the first
trilogy in this series the fact that most of the time it is a military fantasy
makes it interesting also all of the descriptions of food just make me hungry'
```

[9]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_text'].lower() if not c in punct])
    bigrm = nltk.bigrams(r.split())
    for w in bigrm:
        if w in words:
            feat[wordId[w]] +=1
    feat.append(1) #offset
    return feat
```

[10]:
```
X = [feature(d) for d in data]
y = [d['rating'] for d in data]
```

[11]:
```
# Regularized regression
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[12]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[13]:
```
MSE(predictions, y)
```

[13]:
```
1.0178804824879277
```

3. Repeat the above experiment using unigrams and bigrams, still considering the 1000 most common. That is, your model will still use 1000 features (plus an offset), but those 1000 features will be some combination of unigrams and bigrams. Report the MSE obtained using

the new predictor (1 mark).

[14]:
```
words1 = [x[1] for x in counts[:1000]]
wordId1 = dict(zip(words1, range(len(words1))))
wordSet1 = set(words1)
```

[15]:
```
words1_1 = [x for x in counts[:1000]]
words_1 = [x for x in bcounts[:1000]]
biwords = []
for (x,y) in words_1:
    n = y[0] + ' '+ y[1]
    biwords.append((x,n))
```

[76]:
```
new_count = words1_1 + biwords
new_count.sort(reverse = True)
new_count = new_count[:1000]
new_count[:10]
```

[76]:
```
[(73431, 'the'),
 (44301, 'and'),
 (39577, 'a'),
 (36821, 'to'),
 (36581, 'i'),
 (32552, 'of'),
 (21889, 'is'),
 (21468, 'in'),
 (20110, 'it'),
 (19353, 'this')]
```

[17]:
```
new = [x[1] for x in new_count]
newId = dict(zip(new, range(len(new))))
newSet = set(newId)
```

[89]:
```
punct = set(string.punctuation)
def feature1(datum):
    feat = [0]*len(new)
    r = ''.join([c for c in datum['review_text'].lower() if not c in punct])
    bigrm = nltk.bigrams(r.split())
    for w in bigrm:
        b = w[0] +'' + w[1]
        if b in new:
            feat[newId[b]] +=1
    for w in r.split():
        if w in new:
            feat[newId[w]] += 1
    feat.append(1) #offset
    return feat
```

[90]:
```
X1 = [feature1(d) for d in data]
y = [d['rating'] for d in data]
```

[91]:
```
# Regularized regression
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X1, y)
theta1 = clf.coef_
predictions1 = clf.predict(X1)
```

[92]:
```
MSE(predictions1,y)
```

[92]:
```
1.0319906756985915
```

What is the inverse document frequency of the words 'stories', 'magician', 'psychic', 'writing', and 'wonder'? What are their tf-idf scores in the first review (using log base 10, following the first definition of tf-idf given in the slides) (1 mark)?

[22]:
```
nreviews = len(reviews)
word_freq = defaultdict(int)
```

[23]:
```
def feature2(datum):
    w = datum.split()
    w = list(set(w))
    return w
```

[24]:
```
for w in ['stories','magician','psychic','writing','wonder']:
    for r in reviews:
        word_set = feature2(r)
        if w in word_set:
            word_freq[w] += 1
```

[25]:
```
word_freq
```

[25]:
```
defaultdict(int,
            {'stories': 763,
             'magician': 22,
             'psychic': 25,
             'writing': 1005,
             'wonder': 171})
```

[26]:
```
import math
IDFs = defaultdict()
for x in word_freq:
    IDF = math.log(nreviews/word_freq[x],10)
    IDFs[x]=IDF
IDFs
```

[26]:
```
defaultdict(None,
            {'stories': 1.1174754620451195,
             'magician': 2.6575773191777934,
             'psychic': 2.602059991327962,
             'writing': 0.9978339382434922,
             'wonder': 1.7670038896078457})
```

[27]:
```
review1st = reviews[:1]
review1st = review1st[0].split()
```

[28]:
```
TF1streviews=defaultdict(int)
for x in ['stories','magician','psychic','writing','wonder']:
    TF = review1st.count(x)
    TF1streviews[x] = TF
```

[29]:
```
TFIDF1streview=defaultdict(float)

for x in ['stories','magician','psychic','writing','wonder']:
    TFIDF = IDFs[x]*TF1streviews[x]
    TFIDF1streview[x] = TFIDF
TFIDF1streview
```

[29]:
```
defaultdict(float,
            {'stories': 1.1174754620451195,
             'magician': 2.6575773191777934,
             'psychic': 5.204119982655924,
             'writing': 0.9978339382434922,
             'wonder': 1.7670038896078457})
```

5. Adapt your unigram model to use the tfidf scores of words, rather than a bag-of-words representation. That is, rather than your features containing the word counts for the 1000 most common unigrams, it should contain tfidf scores for the 1000 most common unigrams. Report the MSE of this new model (1 mark).

[30]:
```
words2 = defaultdict(int)
for word in words1:
    for r in reviews:
        word_set = feature2(r)
        if word in word_set:
            words2[word] += 1
```

[31]:
```
IDF1000u = defaultdict(float)

for w in words2:
    IDF = math.log(10000/words2[w],10)
    IDF1000u[w] = IDF
```

```
wordId2 = dict(zip(words1, range(len(words1))))
wordSet2 = set(wordId2)
```

[32]:
```
def feature3(datum):
    feat = [0.0]*len(words1)
    wordset = datum.split()
    for w in wordset:
        if w in words1:
            TF = wordset.count(w)
            IDF = IDF1000u[w]
            TFIDF = TF*IDF
            feat[wordId2[w]] = TFIDF
    feat.append(1)
    return feat
```

[33]:
```
X2 = [feature3(r) for r in reviews]
y = [d['rating'] for d in data]
```

[34]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X2, y)
theta2 = clf.coef_
predictions2 = clf.predict(X2)
```

[35]:
```
MSE(predictions2, y)
```

[35]:
```
0.9660150616760584
```

6. Which other review has the highest cosine similarity compared to the first review, in terms of their tf-idf representations (considering unigrams only). Provide the review id, or the text of the review (1 mark)?

[36]:
```
tfidf1st = X2[0]
```

[37]:
```
from scipy import spatial
def cos_simi(datum):
    vetor1 = tfidf1st
    result = 1 - spatial.distance.cosine(vetor1,datum)
    return result
```

[38]:
```
simi = [0]
for x in X2:
    if x == tfidf1st:continue
    cos = cos_simi(x)
    simi.append(cos)
```

[ ]:
```

```

[41]:
```
max1 = max(simi)
max_index = simi.index(max1)
print(max1)
print(data[max_index]['review_id'])
print(data[max_index]['review_text'])
```

```
0.3486253122579983
r81495268
This review is going to be very personal. I will probably not talk about the
writing, the plot or the characters the way that I normally would. Dear Martin
is about 17-year-old Justyce McAllistera a highly successful black student who
attends a primarily all-white private school. One day, he is helping a drunk
friend and arrested. In response, he starts writing letters to Dr, Martin Luther
King Jr. as a way to cope with his newly sharpened awareness of the world around
him
 The world needed this book right now. This is a book that every school needs. I
am going to be buying copies for friends and family. I am a teacher. I can see
how this book would be an excellent way to help students start having these
difficult conversations. Because until we teach young people how to recognize
and talk about racism, sexism, homophobia, and xenophobia and whatever else
needs to get talked about nothing will change. They will unconsciously
internalize the system's biases. They won't see the things that need to change.
And that the things that need to change start with themselves.
 I enjoyed the writing. There are quite a few conversations that are almost
written like a script. Unusual but not distracting and I think that the
convention highlighted the importance of what people were saying. I was fully
investing the characters. So invested that if I don't get regular updates about
Justyce and assurances that he is thriving and well on his way to making the
impact on the world that he deserves I am going to be so mad. Aside from
Justyce, the standouts for me were Manny, SJ, Jared, and Doc.
 One thing that seems to keep coming up is this sense of estrangement from self.
Everyone in this novel (and most likely life) has this idea of who they are that
doesn't quite fit with the reality of who they are. There are these ideas and
visions that other people have about them that they don't know how to reconcile
themselves with. Justyce struggles to reconcile the two sides of his life, Jared
doesn't recognize his internalized racism, and Manny overlooks
 Justyce starts to talk about "the Black Man's Curse" which I hadn't heard about
before. As I understand it, the BMC is the phenomenon that however well a black
man does, however successful he may be they are never able to get to a point
where they don't have to worry about racism. Or about their race affecting their
life. That someone is always waiting to slap you down, tread on your work, and
assume that you don't deserve your success because you are a black man. Just the
short term second-hand frustration of reading about it was overwhelming. I
cannot imagine what it is like to have to live with and deal with it for your
entire life. How strong do you have to be to push through all that is holding
you back? And what does it say about our society that we expect that?
 So read this book. It is well written, it is important, and it is going to stay
```

with you a long time after you finish the final page.

7. Implement a validation pipeline for this same data, by randomly shuffling the data, using 10,000 reviews for training, another 10,000 for validation, and another 10,000 for testing.1 Consider regularization parameters in the range {0.01, 0.1, 1, 10, 100}, and report MSEs on the test set for the model that performs best on the validation set. Using this pipeline, compare the following alternatives in terms of their performance: • Unigrams vs. bigrams • Removing punctuation vs. preserving it. The model that preserves punctuation should treat punctuation characters as separate words, e.g. "Amazing!" would become ['amazing', '!'] • tfidf scores vs. word counts In total you should compare $2 \times 2 \times 2 = 8$ models, and produce a table comparing their performance (2 marks)

[52]:
```
data1 = list(parseData("train_Category.json"))[:30000]
import random
random.shuffle(data1)

train = data1[:10000]
valid = data1[10001:20000]
test = data1[20001:30000]
```

[53]:
```
#Unigrams, remove punt, counts

uniCount = 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():
        uniCount[w] += 1

unicounts = [(uniCount[w], w) for w in uniCount]
unicounts.sort()
unicounts.reverse()

uniwords = [x[1] for x in unicounts[:1000]]
```

[94]:
```
unicounts[:5]
```

[94]:
```
[(74990, 'the'), (45538, 'and'), (39826, 'a'), (37741, 'to'), (36844, 'i')]
```

[54]:
```
### Sentiment analysis

uniwordId = dict(zip(uniwords, range(len(uniwords))))
uniwordSet = set(uniwords)

def feature4(datum):
    feat = [0]*len(uniwords)
    r = ''.join([c for c in datum['review_text'].lower() if not c in␣
 ↪punctuation])
```

```
    for w in r.split():
        if w in uniwords:
            feat[uniwordId[w]] += 1
    feat.append(1) #offset
    return feat
```

[95]:
```
X_univalid = [feature4(d) for d in valid]
y_univalid = [d['rating'] for d in valid]
y_univalid[:5]
```

[95]:
```
[5, 4, 3, 1, 5]
```

[97]:
```
X_univalid[:5]
```

[97]:
```
[[5,
  5,
  3,
  4,
  2,
  1,
  3,
  1,
  1,
  1,
  0,
  1,
  1,
  0,
  1,
  0,
  0,
  0,
  0,
  0,
  0,
  2,
  1,
  1,
  0,
  0,
  0,
  0,
  0,
  1,
  0,
  1,
  0,
```

```
0,
0,
1,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
2,
1,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
```

```
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
2,
```

```
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
```

0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
0,
0,

[99]:
```
mse_univalid = []
for i in [0.01, 0.1, 1, 10, 100]:
            clf = linear_model.Ridge(i, fit_intercept=False) # MSE + 1.0 l2
            clf.fit(X_univalid, y_univalid)
            theta_univalid = clf.coef_
            predictions_univalid = clf.predict(X_univalid)
            mse = MSE(predictions_univalid,y_univalid)
            mse_univalid.append([mse,i])
mse_univalid.sort()
mse_univalid
```

0,

[99]:
```
[[0.970379565822054, 0.01],
 [0.9703796023652271, 0.1],
 [0.9703832038633208, 1],
 [0.9706992980604962, 10],
 [0.9854785856057479, 100]]
```

0,

[101]:
```
#choose 0.01
y_unitest = [d['rating'] for d in test]
X_unitest = [feature4(d) for d in test]
```

0,

[103]:
```
clf = linear_model.Ridge(0.01, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X_unitest, y_unitest)
theta_unitest = clf.coef_
predictions_unitest = clf.predict(X_unitest)
mse_uni = MSE(predictions_unitest,y_unitest)
```

0,

[104]:
```
mse_uni
```

0,

[104]:
```
0.9628687745426 03
```

0,

[105]:
```
#Bigrams, remove punt, counts
biCount = defaultdict(int)
```

0,
0,
0,
0,
0,
0,

```
punctuation = set(string.punctuation)
for d in train:
    r = ''.join([c for c in d['review_text'].lower() if not c in punctuation])
    bigrm = nltk.bigrams(r.split())
    for w in bigrm:
        biCount[w] +=1

bicounts = [(biCount[w], w) for w in biCount]
bicounts.sort()
bicounts.reverse()

biwords = [x[1] for x in bicounts[:1000]]
```

[106]:
```
bicounts[:5]
```

[106]:
```
[(8068, ('of', 'the')),
 (5815, ('in', 'the')),
 (5805, ('this', 'book')),
 (3288, ('the', 'book')),
 (3254, ('and', 'the'))]
```

[109]:
```
biwordId = dict(zip(biwords, range(len(biwords))))
biwordSet = set(biwords)

def feature5(datum):
    feat = [0]*len(biwords)
    r = ''.join([c for c in datum['review_text'].lower() if not c in␣
 ↪punctuation])
    bigrm = nltk.bigrams(r.split())
    for w in bigrm:
        if w in biwords:
            feat[biwordId[w]] +=1
    feat.append(1) #offset
    return feat
```

[110]:
```
X_bivalid = [feature5(d) for d in valid]
y_bivalid = [d['rating'] for d in valid]
y_bivalid[:5]
```

[110]:
```
[5, 4, 3, 1, 5]
```

[111]:
```
mse_bivalid = []
for i in [0.01, 0.1, 1, 10, 100]:
            clf = linear_model.Ridge(i, fit_intercept=False) # MSE + 1.0 l2
            clf.fit(X_bivalid, y_bivalid)
            theta_bivalid = clf.coef_
            predictions_bivalid = clf.predict(X_bivalid)
```

```
            mse = MSE(predictions_bivalid,y_bivalid)
            mse_bivalid.append([mse,i])
mse_bivalid.sort()
mse_bivalid
```

[111]:
```
[[1.0198166879815436, 0.01],
 [1.0198168042373594, 0.1],
 [1.0198274380786303, 1],
 [1.0205454396649567, 10],
 [1.0428920630620062, 100]]
```

[113]:
```
#choose 0.01
y_bitest = [d['rating'] for d in test]
X_bitest = [feature5(d) for d in test]
```

[114]:
```
clf = linear_model.Ridge(0.01, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X_bitest, y_bitest)
theta_bitest = clf.coef_
predictions_bitest = clf.predict(X_bitest)
mse_bi = MSE(predictions_bitest,y_bitest)
mse_bi
```

[114]:
```
1.006064589764447
```

[124]:
```
#unigram, not remove, count
import nltk
nltk.download('punkt')

uniCount1 = defaultdict(int)
for d in train:
    r = ''.join([c for c in d['review_text'].lower()])
    for w in nltk.word_tokenize(r):
        uniCount1[w] += 1

unicounts1 = [(uniCount1[w], w) for w in uniCount1]
unicounts1.sort()
unicounts1.reverse()

uniwords1 = [x[1] for x in unicounts1[:1000]]
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

[127]:
```
unicounts1[:10]
```

[127]:
```
[(81298, '.'),
 (74977, 'the'),
 (69156, ','),
 (45704, 'and'),
 (41493, 'i'),
 (39860, 'a'),
 (37759, 'to'),
 (32579, 'of'),
 (24436, 'it'),
 (22863, 'is')]
```

[132]:
```
uniwordId1 = dict(zip(uniwords1, range(len(uniwords1))))
uniwordSet1 = set(uniwords1)

def feature6(datum):
    feat = [0]*len(uniwords1)
    r = ''.join([c for c in datum['review_text'].lower()])
    for w in nltk.word_tokenize(r):
        if w in uniwords1:
            feat[uniwordId1[w]] += 1
    feat.append(1) #offset
    return feat
```

[133]:
```
X_univalid1 = [feature6(d) for d in valid]
y_univalid1 = [d['rating'] for d in valid]
y_univalid1[:5]
```

[133]:
```
[5, 4, 3, 1, 5]
```

[134]:
```
mse_univalid1 = []
for i in [0.01, 0.1, 1, 10, 100]:
            clf = linear_model.Ridge(i, fit_intercept=False) # MSE + 1.0 l2
            clf.fit(X_univalid1, y_univalid1)
            theta_univalid1 = clf.coef_
            predictions_univalid1 = clf.predict(X_univalid1)
            mse = MSE(predictions_univalid1,y_univalid1)
            mse_univalid1.append([mse,i])

mse_univalid1.sort()
mse_univalid1
```

[134]:
```
[[0.9651208904837766, 0.01],
 [0.9651209274392185, 0.1],
 [0.9651245575409394, 1],
 [0.9654380574447072, 10],
```

```
 [0.9801623070251974, 100]]
```

[135]:
```
#choose 0.01
y_unitest1 = [d['rating'] for d in test]
X_unitest1 = [feature6(d) for d in test]

clf = linear_model.Ridge(0.01, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X_unitest1, y_unitest1)
theta_unitest1 = clf.coef_
predictions_unitest1 = clf.predict(X_unitest1)
mse_uni1 = MSE(predictions_unitest1,y_unitest1)
mse_uni1
```

[135]:
```
0.9639036577816248
```

[145]:
```
#bigram, not remove, count
biCount1 = defaultdict(int)

for d in train:
    r = ''.join([c for c in d['review_text'].lower()])
    r = nltk.word_tokenize(r)
    bigrm = nltk.bigrams(r)
    for w in bigrm:
        biCount1[w] +=1

bicounts1 = [(biCount1[w], w) for w in biCount1]
bicounts1.sort()
bicounts1.reverse()

biwords1 = [x[1] for x in bicounts1[:1000]]
```

[147]:
```
bicounts1[:100]
```

[147]:
```
[(11789, ('.', 'i')),
 (8033, ('of', 'the')),
 (7767, (',', 'and')),
 (6217, ('.', 'the')),
 (6102, (',', 'but')),
 (5829, ('this', 'book')),
 (5774, ('in', 'the')),
 (4530, (',', 'i')),
 (4061, ('.', 'it')),
 (3732, ('it', "'s")),
 (3424, ('it', 'was')),
 (3352, ('the', 'book')),
 (3252, ('and', 'the')),
 (3181, ('is', 'a')),
```

```
(3122, (',', 'the')),
(3057, ('the', 'story')),
(3023, ('i', 'was')),
(2896, ('to', 'the')),
(2894, ('and', 'i')),
(2808, ('to', 'be')),
(2475, ('i', "'m")),
(2443, ('.', 'this')),
(2327, ('book', '.')),
(2270, ('this', 'is')),
(2250, ('did', "n't")),
(2029, ('do', "n't")),
(1969, ('but', 'i')),
(1961, ('with', 'the')),
(1961, ('it', '.')),
(1959, ('in', 'a')),
(1958, ('was', 'a')),
(1911, ('the', 'first')),
(1889, ('for', 'the')),
(1864, ('.', 'but')),
(1819, ('on', 'the')),
(1815, (',', 'it')),
(1804, ('in', 'this')),
(1767, ('it', 'is')),
(1718, ('to', 'read')),
(1704, ('of', 'a')),
(1702, ('that', 'i')),
(1653, ('.', 'and')),
(1646, ('i', 'did')),
(1635, ('the', 'characters')),
(1539, ('.', 'she')),
(1499, ('one', 'of')),
(1482, ('from', 'the')),
(1466, ('at', 'the')),
(1462, ('!', '!')),
(1449, ('is', 'the')),
(1412, ('.', 'there')),
(1393, ('book', ',')),
(1376, ('i', 'have')),
(1364, ('a', 'lot')),
(1348, ('the', 'end')),
(1335, ('i', 'do')),
(1319, ('with', 'a')),
(1317, ('ca', "n't")),
(1315, ('.', 'he')),
(1302, ('a', 'little')),
(1271, ('as', 'a')),
```

```
 (1270, (',', 'a')),
 (1216, ('i', 'would')),
 (1211, ('i', 'really')),
 (1196, ('i', 'think')),
 (1190, ('a', 'bit')),
 (1189, ('of', 'this')),
 (1180, ('i', 'loved')),
 (1175, ('if', 'you')),
 (1161, ('but', 'it')),
 (1157, ('for', 'a')),
 (1151, (')', '.')),
 (1138, ('does', "n't")),
 (1121, ('the', 'series')),
 (1110, ('i', 'am')),
 (1109, ('and', 'it')),
 (1108, ('story', '.')),
 (1092, ('all', 'the')),
 (1078, ('to', 'see')),
 (1062, ('for', 'me')),
 (1060, ('book', 'is')),
 (1056, ('i', "'ve")),
 (1053, ('this', 'one')),
 (1039, ("'s", 'a')),
 (1038, ('was', "n't")),
 (1033, ('this', 'was')),
 (1009, ('the', 'author')),
 (1003, ('and', 'a')),
 (987, ('want', 'to')),
 (986, ('i', 'love')),
 (978, (',', 'this')),
 (974, ('series', '.')),
 (973, ('to', 'get')),
 (970, ('there', 'is')),
 (968, ('that', 'the')),
 (968, ('i', 'had')),
 (965, ('i', 'could')),
 (956, ('this', 'series')),
 (954, ('there', 'are')),
 (951, ('a', 'good'))]
```

[148]:
```
biwordId1 = dict(zip(biwords1, range(len(biwords1))))
biwordSet1 = set(biwords1)

def feature7(datum):
    feat = [0]*len(biwords1)
    r = ''.join([c for c in datum['review_text'].lower()])
    r = nltk.word_tokenize(r)
```

```
    bigrm = nltk.bigrams(r)
    for w in bigrm:
        if w in biwords1:
            feat[biwordId1[w]] +=1
    feat.append(1) #offset
    return feat
```

[149]:
```
X_bivalid1 = [feature7(d) for d in valid]
y_bivalid1 = [d['rating'] for d in valid]
y_bivalid1[:5]
```

[149]:
```
[5, 4, 3, 1, 5]
```

[151]:
```
mse_bivalid1 = []
for i in [0.01, 0.1, 1, 10, 100]:
            clf = linear_model.Ridge(i, fit_intercept=False) # MSE + 1.0 l2
            clf.fit(X_bivalid1, y_bivalid1)
            theta_bivalid1 = clf.coef_
            predictions_bivalid1 = clf.predict(X_bivalid1)
            mse = MSE(predictions_bivalid1,y_bivalid1)
            mse_bivalid1.append([mse,i])
mse_bivalid1.sort()
mse_bivalid1
```

[151]:
```
[[1.00642776066703, 0.01],
 [1.0064278463736203, 0.1],
 [1.006435676436511, 1],
 [1.0069551423361844, 10],
 [1.025238687129851, 100]]
```

[152]:
```
#choose 0.01
y_bitest1 = [d['rating'] for d in test]
X_bitest1 = [feature7(d) for d in test]
clf = linear_model.Ridge(0.01, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X_bitest1, y_bitest1)
theta_bitest1 = clf.coef_
predictions_bitest1 = clf.predict(X_bitest1)
mse_bi1 = MSE(predictions_bitest1,y_bitest1)
mse_bi1
```

[152]:
```
0.9930990285444391
```

[154]:
```
#unigram, remove, tfidf

uniwords_tfidf = defaultdict(int)
for word in uniwords:
    for r in reviews:
```

```
        word_set = feature2(r)
        if word in word_set:
            uniwords_tfidf[word] += 1
```

[156]:
```
uniIDF = defaultdict(float)

for w in uniwords:
    IDF = math.log(10000/uniwords_tfidf[w],10)
    uniIDF[w] = IDF
```

[157]:
```
def feature8(datum):
    feat = [0.0]*len(uniwords)
    wordset = datum.split()
    for w in wordset:
        if w in uniwords:
            TF = wordset.count(w)
            IDF = uniIDF[w]
            TFIDF = TF*IDF
            feat[uniwordId[w]] = TFIDF
    feat.append(1)
    return feat
```

[158]:
```
X_unitest2 = [feature8(r) for r in reviews]
y_unitest2 = [d['rating'] for d in data]
```

[167]:
```
clf = linear_model.Ridge(0.01, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X_unitest2, y_unitest2)
theta_unitest2 = clf.coef_
predictions_unitest2 = clf.predict(X_unitest2)
mse_uni2 = MSE(predictions_unitest2,y_unitest2)
mse_uni2
```

[167]:
```
0.9653503230585385
```

[177]:
```
import pandas as pd
dict = {'MSE_uni_remove_count' : mse_uni,
        'MSE_bi_remove_count' : mse_bi,
       'MSE_uni_notremove_count' : mse_uni1,
       'MSE_bi_notremove_count' : mse_bi1,
       'MSE_uni_remove_tfidf' : mse_uni2
       }
dict
```

[177]:
```
{'MSE_uni_remove_count': 0.962868774542603,
 'MSE_bi_remove_count': 1.006064589764447,
 'MSE_uni_notremove_count': 0.9639036577816248,
```

```
 'MSE_bi_notremove_count': 0.9930990285444391,
 'MSE_uni_remove_tfidf': 0.9653503230585385}
```

[ ]: