Using the code provided on the webpage, read the first 10,000 reviews from the corpus, and read the reviews without capitalization or punctuation.

In [76]:

```
import numpy
from urllib.request import urlopen
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *

def parseData(fname):
  for l in open(fname):
    yield eval(l)

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

In [77]:

```
print("Reading data...")
data = list(parseData("train_Category.json"))[:10000]
print("done")
```

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

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).

In [3]:

```
review = [k['review_text'] for k in data]
```

Ignore capitalization

In [4]:

```
review_lower = [k.lower() for k in review]
```

Remove punctuation

In [5]:

```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for k in review_lower:
  r = ''.join([c for c in k if not c in punctuation])
  for w in r.split():
    wordCount[w] += 1
```

In [6]:

```
# total number of words without punctuation and capital letter
print('Total number of single words',len(wordCount))
```

```
Total number of single words 73286
```

In [7]:

```
bigramCount = defaultdict(int)
punctuation = set(string.punctuation)
for k in review_lower:
  r = ''.join([c for c in k if not c in punctuation])
  for m in range(len(r.split())-1):
    bigram = r.split()[m] +' '+ r.split()[m+1]
    bigramCount[bigram] += 1
```

In [8]:

```
counts = [(bigramCount[b], b) for b in bigramCount]
counts.sort()
counts.reverse()
```

Answer for question 1:

In [9]:

```
print('Total number of bigrams', len(bigramCount))
```

```
Total number of bigrams 521502
```

In [10]:

```
print(counts[:5])
```

```
[(7927, 'of the'), (5850, 'this book'), (5627, 'in the'), (3189, 'an
d 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.

In [11]:

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

In [12]:

```
bigrams[:10]
```

Out[12]:

```
['of the',
 'this book',
 'in the',
 'and the',
 'is a',
 'the book',
 'it was',
 'the story',
 'to the',
 'i was']
```

In [13]:

```
bigramSet = set(bigrams)
```

In [14]:

```
len(bigramSet)
```

Out[14]:

```
1000
```

In [15]:

```
y = [d['rating'] for d in data]
```

In [16]:

```
bigramId = dict(zip(bigrams, range(len(bigrams))))
```

In [17]:

```
len(bigramId)
# bigramId is a dictionary with the form (biagram, index from their rank) with 1
000 entry
```

Out[17]:

```
1000
```

In [18]:

```
def feature(datum):
  feat = [0]*len(bigrams)
  r = ''.join([c for c in datum['review_text'].lower() if not c in punctuation])
  for m in range(len(r.split())-1):
    bigram = r.split()[m] +' '+ r.split()[m+1]
    if bigram in bigrams:
      feat[bigramId[bigram]] += 1 # feat[index of the biagram]
  feat.append(1) #offset
  return feat
```

In [19]:

```
X = [feature(d) for d in data]
# d is each entry in our data set
# for each d, we extract the review_text, put it into lower case and remomve pun
ctuation.
# then for each review, we find whether those bigrams are in the bigrams set, wh
ich is the most common 1000 bigrams
```

In [20]:

```
theta,residuals,rank,s = numpy.linalg.lstsq(X, y)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  """Entry point for launching an IPython kernel.
```

In [21]:

```
print(residuals)
```

```
[10178.68026215]
```

In [22]:

```
mseBigram = residuals/len(y)
```

In [23]:

```
mseBigram[0]
```

Out[23]:

```
1.0178680262146527
```

Answer for question 2:

In [24]:

```
print('MSE for the bigram model:',mseBigram[0])
```

```
MSE for the bigram model: 1.0178680262146527
```

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).

In [25]:

```
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

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

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

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

In [26]:

```
counts.extend(countsword)
```

In [27]:

```
counts.sort()
counts.reverse()
print(counts[100:150])
```

```
[(2169, 'know'), (2154, 'which'), (2149, 'this is'), (2145, 'do'),
(2142, 'dont'), (2133, 'didnt'), (2109, 'did'), (2101, 'only'), (206
2, 'we'), (2054, 'character'), (2044, 'little'), (2033, 'loved'), (2
023, 'world'), (1986, 'was a'), (1967, 'life'), (1946, 'in a'), (192
9, 'too'), (1924, 'for the'), (1905, 'see'), (1871, 'end'), (1856,
'with the'), (1837, 'could'), (1814, 'the first'), (1814, 'but i'),
(1804, 'on the'), (1784, 'novel'), (1771, 'still'), (1756, 'in thi
s'), (1748, 'new'), (1740, 'after'), (1713, 'going'), (1702, 'thing
s'), (1690, 'to read'), (1685, 'it is'), (1671, 'of a'), (1668, 'the
n'), (1665, 'de'), (1662, 'being'), (1647, 'though'), (1625, 'the ch
aracters'), (1613, 'two'), (1607, 'through'), (1588, 'while'), (156
2, 'people'), (1539, 'author'), (1536, 'bit'), (1523, 'that i'), (15
21, 'where'), (1512, 'e'), (1508, 'one of')]
```

In [28]:

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

In [29]:

```
UBId = dict(zip(ubmostcom, range(len(ubmostcom))))
```

In [30]:

```
len(UBId)
```

Out[30]:

```
1000
```

In [31]:

```
def featureUB(datum):
  feat = [0]*len(UBId)
  r = ''.join([c for c in datum['review_text'].lower() if not c in punctuation])
  for w in r.split():
    if w in UBId:
      feat[UBId[w]] += 1
  for m in range(len(r.split())-1):
    bigram = r.split()[m] +' '+ r.split()[m+1]
    if bigram in UBId:
      feat[UBId[bigram]] += 1 # feat[index of the biagram]
  feat.append(1) #offset
  return feat
```

In [32]:

```
XUB = [featureUB(d) for d in data]
```

In [33]:

```
y = [d['rating'] for d in data]
```

In [34]:

```
thetaUB,residualsUB,rankUB,sUB = numpy.linalg.lstsq(XUB, y)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  """Entry point for launching an IPython kernel.
```

In [35]:

```
residualsUB
```

Out[35]:

```
array([9683.58022078])
```

In [36]:

```
mseUB = residualsUB/len(y)
```

In [37]:

```
mseUB[0]
```

Out[37]:

```
0.9683580220775222
```

Answer for question 3:

In [38]:

```
print('MSE for the Unigram and Bigram model:',mseUB[0])
```

```
MSE for the Unigram and Bigram model: 0.9683580220775222
```

4. 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) (1 mark)?

Inverse document frequency - How “rare” is this term across all documents

In [39]:

```
# the first review:
review_lower[0]
```

Out[39]:

```
"genuinely enthralling. if collins or bernard did invent this out of
whole cloth, they deserve a medal for imagination. lets leave the ve
racity aside for a moment - always a touchy subject when it comes to
real life stories of the occult - and talk about the contents. \n th
e black alchemist covers a period of two years in which collins, a m
agician, and bernard, a psychic, undertook a series of psychic quest
s that put them in opposition with the titular black alchemist. as e
ntertainment goes, the combination of harrowing discoveries, ancient
lore, and going down the pub for a cigarette and a guinness, trying
to make sense of it all while a hen party screams at each other, is
a winner. it is simultaneously down to earth and out of this world.
\n it reads fast, both because of the curiousity and because collins
has a very clear writing style. sometimes its a little clunky or ove
r repetitive and there's a few meetings that get underreported, but
i am very much quibbling here. mostly important, he captures his own
and bernard's sense of wonder, awe and occasionally revulsion enough
that i shared them."
```

In [61]:

```
r = ''.join([c for c in review_lower[0] if c not in punctuation])
r
```

Out[61]:

```
'genuinely enthralling if collins or bernard did invent this out of
whole cloth they deserve a medal for imagination lets leave the vera
city aside for a moment  always a touchy subject when it comes to re
al life stories of the occult  and talk about the contents \n the bl
ack alchemist covers a period of two years in which collins a magici
an and bernard a psychic undertook a series of psychic quests that p
ut them in opposition with the titular black alchemist as entertainm
ent goes the combination of harrowing discoveries ancient lore and g
oing down the pub for a cigarette and a guinness trying to make sens
e of it all while a hen party screams at each other is a winner it i
s simultaneously down to earth and out of this world \n it reads fas
t both because of the curiousity and because collins has a very clea
r writing style sometimes its a little clunky or over repetitive and
theres a few meetings that get underreported but i am very much quib
bling here mostly important he captures his own and bernards sense o
f wonder awe and occasionally revulsion enough that i shared them'
```

In [64]:

```
# term frequency in the first review:
goal = {'stories','magician', 'psychic', 'writing', 'wonder'}
tf = defaultdict(int)
for w in r.split():
    if w in goal:
      tf[w] += 1
```

In [65]:

```
# tf in the first review
tf
```

Out[65]:

```
defaultdict(int,
            {'stories': 1,
             'magician': 1,
             'psychic': 2,
             'writing': 1,
             'wonder': 1})
```

In [66]:

```
def goalAppearsTimesDetect(goalWord):
    count = 0
    for d in review_lower:
        r = ''.join([c for c in d if not c in punctuation])
        if goalWord in r.split(): count += 1
    return count
```

In [67]:

```
goalAppearsTimesDetect('stories')
```

Out[67]:

```
763
```

In [68]:

```
df = defaultdict(int)
```

In [69]:

```
for g in goal:
    df[g] = goalAppearsTimesDetect(g)
```

In [70]:

```
df
```

Out[70]:

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

In [71]:

```
idf = defaultdict(float)
```

In [72]:

```
import math
for g in goal:
    idf[g] = math.log10(len(review_lower)/df[g])
```

Answer for question 4:

In [73]:

```
# idf for the target words
for g in idf:
    print('The idf for',g,'is:', idf[g])
```

```
The idf for wonder is: 1.7670038896078462
The idf for stories is: 1.1174754620451195
The idf for writing is: 0.9978339382434923
The idf for psychic is: 2.6020599913279625
The idf for magician is: 2.657577319177794
```

In [74]:

```
tfidf = defaultdict(float)
```

In [75]:

```
for g in goal:
    tfidf[g] = tf[g] * idf[g]
```

In [76]:

```
# tfidf for the target words in the first review
for g in tfidf:
    print('The tfidf for',g,'in the first review is:', tfidf[g])
```

```
The tfidf for wonder in the first review is: 1.7670038896078462
The tfidf for stories in the first review is: 1.1174754620451195
The tfidf for writing in the first review is: 0.9978339382434923
The tfidf for psychic in the first review is: 5.204119982655925
The tfidf for magician in the first review is: 2.657577319177794
```

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).

In [77]:

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

In [79]:

```
dfWords = defaultdict(int)
```

In [84]:

```
for g in words:
    dfWords[g] = goalAppearsTimesDetect(g)
```

In [85]:

```
def tfidfCal(reviewEntry):
    r = ''.join([c for c in reviewEntry if not c in punctuation])
    tf = defaultdict(int)
    feat = [0]*len(wordSet)

    for w in r.split():
        if w in wordSet:
            tf[w] += 1

    idf = defaultdict(float)
    tfidf = defaultdict(float)
    for g in wordSet:
        idf[g] = math.log10(len(review_lower)/dfWords[g])
        tfidf[g] = tf[g] * idf[g]
        feat[wordId[g]] = tfidf[g]

    feat.append(1) # offset
    return feat
```

In [86]:

```
X_tfidf = [tfidfCal(d) for d in review_lower]
```

In [87]:

```
len(X_tfidf)
```

Out[87]:

```
10000
```

In [88]:

```
len(X_tfidf[0])
```

Out[88]:

```
1001
```

In [89]:

```
y = [d['rating'] for d in data]
```

In [90]:

```
thetai,residualsi,ranki,si = numpy.linalg.lstsq(X_tfidf, y)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  """Entry point for launching an IPython kernel.
```

In [91]:

```
residualsi
```

Out[91]:

```
array([9660.14227269])
```

In [92]:

```
msei = residualsi/len(y)
```

Answer for question 5:

In [93]:

```
print('MSE for the Unigram tfidf model:',msei[0])
```

```
MSE for the Unigram tfidf model: 0.9660142272690154
```

6. Which other review has the highest cosine similarity compared to the first review (provide the review id, or the text of the review) (1 mark)?

In [94]:

```
def consineSim(a,b):
    return numpy.dot(a,b)/(numpy.linalg.norm(a)*numpy.linalg.norm(b))
```

In [95]:

```
simMax = []
for k in X_tfidf[1:]:
    simMax.append(consineSim(X_tfidf[0],k))
```

In [96]:

```
max(simMax)
indexOfMaxSim = simMax.index(max(simMax)) + 1
```

In [97]:

```
print('The maximum cosine similarity among all the reviews compared to the first
review is'
      ,max(simMax),'with the index of review',indexOfMaxSim)
```

```
The maximum cosine similarity among all the reviews compared to the
first review is 0.34862531225799825 with the index of review 6921
```

Answer for question 6:

In [98]:

```
#review = [k['review_text'] for k in data]
reviweId = data[6921]['review_id']
print('The review id is for the most similar review is:',reviweId )
```

```
The review id is for the most similar review is: r81495268
```

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 punc- tuation characters as separate words, e.g. “Amazing!” would become [‘amazing’, ‘!’]

• tfidf scores vs. word counts In total you should compare 2 × 2 × 2 = 8 models, and produce a table comparing their performance (2 marks)

In [78]:

```
from sklearn.utils import shuffle
```

In [79]:

```
data7 = list(parseData("train_Category.json"))
```

In [80]:

```
random.shuffle(data7)
```

In [81]:

```
review7 = [k['review_text'] for k in data7[:30000]]
```

In [82]:

```
reviewTrain = review7[:10000]
reviewTrain = [k.lower() for k in reviewTrain]
```

In [83]:

```
reviewVal = review7[10000:20000]
reviewVal = [k.lower() for k in reviewVal]
```

In [84]:

```
reviewTest =  review7[20000:]
reviewTest = [k.lower() for k in reviewTest]
```

In [85]:

```
reg = [0.001, 0.1, 1, 10,100]
```

In [86]:

```
rating7 = [k['rating'] for k in data7]
```

In [87]:

```
yTrain = rating7[:10000]
```

In [88]:

```
yVal = rating7[10000:20000]
```

In [89]:

```
yTest =  rating7[20000:]
```

In [90]:

```
#Function that could give us the words set with unigram/bigram and with/without
 punctuation
def isUnigram(is_Uni,remove_Punc,reviewList):
    punctuation = set(string.punctuation)
    if remove_Punc: # remove punctuation
        wordCount = defaultdict(int)
        for k in reviewList:
            r = ''.join([c for c in k if not c in punctuation])
            if is_Uni: # unigram
                for w in r.split():
                    wordCount[w] += 1
            else: # bigram
                for m in range(len(r.split())-1):
                    w = r.split()[m] +' '+ r.split()[m+1]
                    wordCount[w] += 1

        countsword = [(wordCount[w], w) for w in wordCount]
        countsword.sort()
        countsword.reverse()
        # return the words set without punctuation
        words = [x[1] for x in countsword[:1000]]
        wordId = dict(zip(words, range(len(words))))
        wordSet = set(words)

    else: # do not remove punctuation
        wordCount = defaultdict(int)
        puntCount = defaultdict(int)
        for k in reviewList:
            r = ''.join([c for c in k if not c in punctuation])
            rp = ''.join([c for c in k if c in punctuation])
            if is_Uni: # punct with Uni
                for w in r.split():
                    wordCount[w] += 1
                for p in rp:
                    puntCount[p] += 1
            else: # punt with Bi
                for m in range(len(r.split())-1):
                    w = r.split()[m] +' '+ r.split()[m+1]
                    wordCount[w] += 1
                for p in rp:
                    puntCount[p] += 1

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

        countsPunc = [(puntCount[p], p) for p in puntCount]
        countsPunc.sort()
        countsPunc.reverse()

        countsword.extend(countsPunc)
        countsword.sort()
        countsword.reverse()
        # return the words set with punctuation
        words = [x[1] for x in countsword[:1000]]
        wordId = dict(zip(words, range(len(words))))
        wordSet = set(words)

    return words,wordId,wordSet,countsword
```

word counts, Unigram, remove punctuation

In [112]:

```
words,wordId,wordSet,countsword = isUnigram(True,True,reviewTrain)
words1,wordId1,wordSet1,countsword1 = isUnigram(True,True,reviewVal)
words2,wordId2,wordSet2,countsword2 = isUnigram(True,True,reviewTest)
```

word counts, Bigram, remove punctuation

In [113]:

```
wordsb,wordIdb,wordSetb,countswordb = isUnigram(False,True,reviewTrain)
wordsb1,wordIdb1,wordSetb1,countswordb1 = isUnigram(False,True,reviewVal)
wordsb2,wordIdb2,wordSetb2,countswordb2 = isUnigram(False,True,reviewTest)
```

word counts, Unigram, with punctuation

In [114]:

```
wordsp,wordIdp,wordSetp,countswordp = isUnigram(True,False,reviewTrain)
wordsp1,wordIdp1,wordSetp1,countswordp1 = isUnigram(True,False,reviewVal)
wordsp2,wordIdp2,wordSetp2,countswordp2 = isUnigram(True,False,reviewTest)
```

word counts, Bigram, with punctuation

In [115]:

```
wordspp,wordIdpp,wordSetpp,countswordpp = isUnigram(False,False,reviewTrain)
wordsp1p,wordIdp1p,wordSetp1p,countswordp1p = isUnigram(False,False,reviewVal)
wordsp2p,wordIdp2p,wordSetp2p,countswordp2p = isUnigram(False,False,reviewTest)
```

Words Count

In [116]:

```
from sklearn import linear_model
punctuation = set(string.punctuation)
```

In [117]:

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

In [118]:

```
def feature_X(review,words,wordId,wordSet,is_Uni):
    feat = [0]*len(words)
    r = ''.join([c for c in review if c not in punctuation])
    rp = ''.join([c for c in review if c in punctuation])

    if is_Uni:
      for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    else:
      for m in range(len(r.split())-1):
        w = r.split()[m] +' '+ r.split()[m+1]
        if w in words:
            feat[wordId[w]] += 1

    for p in rp:
        if p in words:
            feat[wordId[p]] += 1

    feat.append(1)
    return feat
```

In [119]:

```
def modelTestAutoWordsCount(is_Uni,remove_Punc):
    # Validation Set
    words,wordId,wordSet,countsword = isUnigram(is_Uni,remove_Punc,reviewVal)
    XVal = [feature_X(r,words,wordId,wordSet,is_Uni) for r in reviewVal]

    words1,wordId1,wordSet1,countsword1 = isUnigram(is_Uni,remove_Punc,reviewTra
in)
    XTrain = [feature_X(r,words1,wordId1,wordSet1,is_Uni) for r in reviewTrain]

    words2,wordId2,wordSet2,countsword2 = isUnigram(is_Uni,remove_Punc,reviewTes
t)
    XTest = [feature_X(r,words2,wordId2,wordSet2,is_Uni) for r in reviewTest]

    mseMin = 100
    ind = 0
    for a in reg:
        clf = linear_model.Ridge(a, fit_intercept=False) # MSE
        clf.fit(XTrain, yTrain)
        predictionsVal = clf.predict(XVal)
        if MSE(predictionsVal,yVal) <= mseMin:
            mseMin = MSE(predictionsVal,yVal)
            ind = a

    print('MSE for word counts with reg',ind,'is the minimum on the Validation s
et with', mseMin)
    clf = linear_model.Ridge(ind, fit_intercept=False) # MSE
    clf.fit(XTrain, yTrain)
    predictionsTest = clf.predict(XTest)
    print('MSE for the test data',ind,'is', MSE(predictionsTest,yTest))
```

word counts, Unigram, remove punctuation

In [141]:

```
modelTestAutoWordsCount(True,True)
```

```
MSE for wordCount model with reg 100 is the minimum on the Validatio
n set with 1.1521957460385754
MSE for wordCount model on the test data 100 is 1.178301038264732134
33
```

word counts, Bigram, remove punctuation

In [142]:

```
modelTestAutoWordsCount(False,True)
```

```
MSE for wordCount model with reg 100 is the minimum on the Validatio
n set with 1.1841482772280012
MSE for wordCount model on the test data 100 is 1.199638103928011231
12
```

word counts, Unigram, with punctuation

In [143]:

```
modelTestAutoWordsCount(True,False)
```

```
MSE for wordCount model with reg 100 is the minimum on the Validatio
n set with 1.1730038274625409
MSE for wordCount model on the test data 100 is 1.199503958375173849
11
```

word counts, Bigram, with punctuation

In [144]:

```
modelTestAutoWordsCount(False,False)
```

```
MSE for wordCount model with reg 100 is the minimum on the Validatio
n set with 1.1916029374619102
MSE for wordCount model on the test data 100 is 1.202143241947100413
74
```

tfidf Scores

In [124]:

```
def goalAppearsTimesDetect(goalWord,is_Uni,remove_Punc,reviewList):
    count = 0
    for d in reviewList:
        notAppear = True
        #print('enter the list',reviewList.index(d))
        r = ''.join([c for c in d if not c in punctuation])
        rp = ''.join([c for c in d if c in punctuation])

        if is_Uni:
            if goalWord in r.split(): count += 1

        else:
            for m in range(len(r.split())-1):
                w = r.split()[m] +' '+ r.split()[m+1]
                if w == goalWord:
                    count += 1
                    break

        if not remove_Punc:
            if p in rp: count += 1
    return count
```

In [134]:

```
def goalAppearsTimesDetectSet(goalSet,is_Uni,remove_Punc,reviewList):
    count = defaultdict(int)
    for d in reviewList:
        r = ''.join([c for c in d if not c in punctuation])
        rp = ''.join([c for c in d if c in punctuation])

        if is_Uni:
            for w in r.split():
                if w in goalSet:
                    count[w] += 1
                    break

        else:
            for m in range(len(r.split())-1):
                w = r.split()[m] +' '+ r.split()[m+1]
                if w in goalSet:
                    count[w] += 1
                    break

        if not remove_Punc:
            for p in rp:
                if p in goalSet:
                    count[p] += 1
                    break
    return count
```

In [129]:

```
def feature_X_tfidf(review,words,wordId,wordSet,is_Uni,remove_Punc,reviewList,id
f):
    feat = [0]*len(words)
    tf = defaultdict(int)
    tfidf = defaultdict(float)

    r = ''.join([c for c in review if not c in punctuation])
    rp = ''.join([c for c in review if c in punctuation])

    if is_Uni:
        for w in r.split():
            if w in words:
                tf[w] += 1
    else:
        for m in range(len(r.split())-1):
            w = r.split()[m] +' '+ r.split()[m+1]
            if w in words:
                tf[w] += 1

    if not remove_Punc:
        for p in rp:
            if p in words:
                tf[w] += 1

    for g in wordSet:
        #print(g,'X_tfidf')
        tfidf[g] = tf[g] * idf[g]
        feat[wordId[g]] = tfidf[g]
    feat.append(1)
    return feat
```

In [130]:

```
def modelTestAutoWordsCountTf(XVal,XTrain,XTest):
    mseMin = 100
    ind = 0
    for a in reg:
        clf = linear_model.Ridge(a, fit_intercept=False) # MSE
        clf.fit(XTrain, yTrain)
        predictionsVal = clf.predict(XVal)
        print(a,MSE(predictionsVal,yVal))
        if MSE(predictionsVal,yVal) <= mseMin:
            mseMin = MSE(predictionsVal,yVal)
            ind = a
    print('MSE for tfidf model with reg',ind,'is the minimum on the Validation s
et with', mseMin)

    # Test Set
    clf = linear_model.Ridge(ind, fit_intercept=False) # MSE
    clf.fit(XTrain, yTrain)
    predictionsTest = clf.predict(XTest)
    print('MSE for tfidf model on the test data',ind,'is', MSE(predictionsTest,y
Test))
```

tfidf, Unigram, remove punctuation

In [146]:

```
def prepareX(is_Uni,remove_Punc,reviewList,words,wordId,wordSet,countsword):
    dfWords = defaultdict(int)
    idf = defaultdict(float)
    dfWords = goalAppearsTimesDetectSet(wordSet,is_Uni,remove_Punc,reviewList)
    for g in dfWords:
        idf[g] = math.log10(10000/dfWords[g])
    print('df,idf ready')
    X = [feature_X_tfidf(r,words,wordId,wordSet,is_Uni,remove_Punc,reviewList,id
f) for r in reviewList]
    return X
```

In [156]:

```
def prepareX(is_Uni,remove_Punc,reviewList,words,wordId,wordSet,countsword):
    return
```

In [157]:

```
# Validation set
XVal = prepareX(True,True,reviewVal,words,wordId,wordSet,countsword)
print('Val done')
# Train set
XTrain = prepareX(True,True,reviewTrain,words1,wordId1,wordSet1,countsword1 )
print('Train done')
# Test set
XTest = prepareX(True,True,reviewTest,words2,wordId2,wordSet2,countsword2)
print('Test done')
```

```
Val done
Train done
Test done
```

In [158]:

```
modelTestAutoWordsCountTf(XVal,XTrain,XTest)
```

```
MSE for tfidf model with reg 100 is the minimum on the Validation se
t with 1.1615343572592211
MSE for tfidf model on the test data 100 is 1.17940103826473213433
```

tfidf, Bigram, remove punctuation

In [160]:

```
# Validation set
XVal1 = prepareX(True,True,reviewVal,wordsb,wordIdb,wordSetb,countswordb)
print('Val done')
# Train set
XTrain1 = prepareX(True,True,reviewTrain,wordsb1,wordIdb1,wordSetb1,countswordb1
)
print('Train done')
# Test set
XTest1 = prepareX(True,True,reviewTest,wordsb2,wordIdb2,wordSetb2,countswordb2)
print('Test done')
```

```
Val done
Train done
Test done
```

In [161]:

```
modelTestAutoWordsCountTf(XVal1,XTrain1,XTest1)
```

```
MSE for tfidf model with reg 100 is the minimum on the Validation se
t with 1.2071482772280012
MSE for tfidf model on the test data 100 is 1.22633815039483723112
```

tfidf, Unigram, with punctuation

In [163]:

```
# Validation set
XVal2 = prepareX(True,False,reviewVal,wordsp,wordIdp,wordSetp,countswordp)
print('Val done')
# Train set
XTrain2 = prepareX(True,False,reviewTrain,wordsp1,wordIdp1,wordSetp1,countswordp
1 )
print('Train done')
# Test set
XTest2 = prepareX(True,False,reviewTest,wordsp2,wordIdp2,wordSetp2,countswordp2)
print('Test done')
```

```
Val done
Train done
Test done
```

In [164]:

```
modelTestAutoWordsCountTf(XVal2,XTrain2,XTest2)
```

```
MSE for tfidf model with reg 100 is the minimum on the Validation se
t with 1.1823038241325409
MSE for tfidf model on the test data 100 is 1.19580395828117384911
```

tfidf, Bigram, with punctuation

In [166]:

```
# Validation set
XVal3 = prepareX(False,False,reviewVal,wordspp,wordIdpp,wordSetpp,countswordpp)
print('Val done')
# Train set
XTrain3 = prepareX(False,False,reviewTrain,wordsp1p,wordIdp1p,wordSetp1p,countsw
ordp1p)
print('Train done')
# Test set
XTest3 = prepareX(False,False,reviewTest,wordsp2p,wordIdp2p,wordSetp2p,countswor
dp2p)
print('Test done')
```

```
Val done
Train done
Test done
```

In [167]:

```
modelTestAutoWordsCountTf(XVal3,XTrain3,XTest3)
```

```
MSE for tfidf model with reg 100 is the minimum on the Validation se
t with 1.2198029370294103
MSE for tfidf model on the test data 100 is 1.22251220394710041374
```

Answer for question 7:

In [168]:

```
print('The best model we have is the word count model with Unigram and Punctuati
on removed')
```

```
The best model we have is the word count model with Unigram and Punc
tuation removed
```

In [170]:

```
d = {1: ['Unigram no Punctuation',1.1783, 1.1794],
     2: ['Bigram no Punctuation', 1.1996, 1.2263],
     3: ['Unigram with Punctuation', 1.1995, 1.1958],
     4: ['Bigram with Punctuation', 1.2021, 1.2225]}
print('Test MSE on different model')
print ("{:<8} {:<30} {:<20} {:<20}".format('ind','Performance','Word Count','tfi
df'))
for k, v in d.items():
    lang, perc, change = v
    print ("{:<8} {:<30} {:<20} {:<20}".format(k, lang, perc, change))
```

```
Test MSE on different model
ind      Performance                    Word Count           tfidf
1        Unigram no Punctuation         1.1783               1.1794
2        Bigram no Punctuation          1.1996               1.2263
3        Unigram with Punctuation       1.1995               1.1958
4        Bigram with Punctuation        1.2021               1.2225
```