Problem1:
The method I use to calculate unique bigram is for every review, I find mostly '\t' separate the sentences in one review. So after removing the punctuation and capitalization, I use split('\t') to divide the whole review into several sub-sentences, and I find the unique bigrams among these sub-sentences and regard these as unique bigram for this review. By doing this, my answer is:

the number of unique bigrams are: 176685
the 5 most frequently-occurring bigrams are :
(('with', 'a'), 4586)
(('in', 'the'), 2593)
(('of', 'the'), 2243)
(('is', 'a'), 2053)
(('on', 'the'), 2031)

Problem2:
MSE for using 1000 most common bigrams is: 1.16956086931e-05

Problem3:
I choose 50%-50% of most common unigrams and bigrams
MSE for using 1000 combination feature is: 4.16148936278e-05

Problem4:
5 unigrams/bigrams with most negative associated weights:
unigram/bigram is : ('at', 'a') weight is : -0.258264193984
unigram/bigram is : ('a', 'pale') weight is : -0.212990993127
unigram/bigram is : ('tastes', 'like') weight is : -0.205865954323
unigram/bigram is : ('sort', 'of') weight is : -0.190997816898
unigram/bigram is : ('the', 'bitterness') weight is : -0.177478809295

5 unigrams/bigrams with most positive associated weights:
unigram/bigram is : ('very', 'drinkable') weight is : 0.286631586356

unigram/bigram is : ('i', 'love') weight is : 0.253146868245
unigram/bigram is : ('the', 'best') weight is : 0.241321052988
unigram/bigram is : ('easy', 'to') weight is : 0.231415813322
unigram/bigram is : ('up', 'a') weight is : 0.203191599824

Problem5:
the idf of word foam is: 1.1378686206869628
the idf of word smell is: 0.5379016188648442
the idf of word banana is: 1.6777807052660807
the idf of word lactic is: 2.9208187539523753
the idf of word tart is: 1.8068754016455384
tf-idf score of word foam in the first review is : 2.2757372413739256
tf-idf score of word smell in the first review is : 0.5379016188648442
tf-idf score of word banana in the first review is : 3.3555614105321614
tf-idf score of word lactic in the first review is : 5.841637507904751
tf-idf score of word tart in the first review is : 1.8068754016455384

Problem6:
cos_theta = 0.106130241679 theta = 0.466153952679 pi

Problem7:
'review/text': 'Poured from a 22oz bottle to a Dogfish Head Snifter.\t\tC
olor: Slight hazy orange with an off white head.\t\tSmell: Cinnamon, ba
nana, pumpkin and nutmeg.\t\tTaste: Alcohol, pumpkin, nutmeg, allspi
ce and a hint of banana.\t\tMouthfeel: Medium carbonation, smooth,
medium dryness on the palate.\t\tOverall: The smell is GREAT! The ban
ana was a huge surprise for me. The taste had too much alcohol presen
ce. Seemed to overpower the other flavors. Cheers!',

Problem8:
MSE for tfidf model is: 1.25765745218e-05

Code:

```
import numpy
import urllib.request
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
def parseData(fname):
    for l in urllib.request.urlopen(fname):
        yield eval(l)
print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse258/data/beer/beer_50000.json"))[:5000]
print ("done")
```

```
#problem 1
punctuation = set(string.punctuation)
review = []
for d in data:
    per_review = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    review.append(per_review.split('\t'))
sentence = []
for d in review:
    for i in d:
        if i != '' and len(i)>1:
            sentence.append(i)
bigrams = [b for d in sentence for b in zip(d.split(' ')[:-1],d.split(' ')[1:])]
bigrams_count = defaultdict(int)
for i in bigrams:
    bigrams_count[i] += 1
import operator
sorted_bc = sorted(bigrams_count.items(),key = operator.itemgetter(1),reverse = True)
print('the number of unique bigrams are: ',len(sorted_bc))
print('the 5 most frequently-occurring bigrams are :' )
for i in range(5):
      print(sorted_bc[i])
```

```
#or sorting using this code:
counts = [(bigrams_count[w],w) for w in bigrams_count]
counts.sort()
counts.reverse()
```

```
#problem 2
bigram_1000 = [x[1] for x in counts[:1000]]
wordID = dict(zip(bigram_1000,range(len(bigram_1000))))
def feature(datum):
    feat = [0]*len(bigram_1000)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
    review_without_t = r.split('\t')
    sentence1 = []
    for i in review_without_t:
        if i != '' and len(i)>1:
            sentence1.append(i)
    bigram_sentence = [b for d in sentence1 for b in zip(d.split(' ')[:-1],d.split(' ')[1:])]
    for b in bigram_sentence:
        if b in bigram_1000:
            feat[wordID[b]] +=1
    feat.append(1)
    return feat
```

```
X = [feature(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)
MSE = 0
for i in range(len(y)):
    MSE = (y[i]-predictions[i])**2
MSE = MSE/len(y)
print('MSE for using 1000 most common bigrams is: ',MSE)
```

```
#problem 3
#using 50% most popular in unigram model and 50% in bigram model
wordCount = defaultdict(int)
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_uni = [(wordCount[w],w) for w in wordCount]
counts_uni.sort()
counts_uni.reverse()
unigram_1000 = [x[1] for x in counts_uni[:1000]]
```

```
new_popular = unigram_1000[:500]+bigram_1000[:500]
new_wordID = dict(zip(new_popular,range(len(bigram_1000))))
def newfeature(datum):
    feat = [0]*len(new_wordID)
    #deal with unigram
    ru = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in ru.split():
        if w in new_popular:
            feat[new_wordID[w]] +=1
    #deal with bigram
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
    review_without_t = r.split('\t')
    sentence1 = []
    for i in review_without_t:
        if i != '' and len(i)>1:
            sentence1.append(i)
    bigram_sentence = [b for d in sentence1 for b in zip(d.split(' ')[:-1],d.split(' ')[1:])]
    for b in bigram_sentence:
        if b in new_popular:
            feat[new_wordID[b]] +=1
    feat.append(1)
    return feat
```

```
X = [newfeature(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)
MSE = 0
for i in range(len(y)):
    MSE = (y[i]-predictions[i])**2
MSE = MSE/len(y)
print('MSE for using 1000 combination feature is: ',MSE)
```

```
#problem 4
#0-499 unigram model   #500-999 bigram model
theta = list(theta)
index = list(zip(theta,new_popular1))
sorted_index = sorted(index, key=lambda tup: tup[0])
```

```
print('5 unigrams/bigrams with most negative associated weights: ')
for i in range(5):
    print('unigram/bigram is :',sorted_index[i][1],'weight is :',sorted_index[i][0] )
```

```
sorted_again = sorted(index, key=lambda tup: tup[0],reverse = True)
print('5 unigrams/bigrams with most positive associated weights: ')
for i in range(5):
    print('unigram/bigram is :',sorted_again[i][1],'weight is :',sorted_again[i][0] )
```

```
#problem 5
def count(w):
    final_count = 0
    for d in data:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
        if w in r.split():
            final_count += 1
    return final_count
```

```
from math import log10
N=5000
df_foam = count('foam')
df_smell = count('smell')
df_banana = count('banana')
df_lactic = count('lactic')
df_tart = count('tart')
idf_foam = log10(N/df_foam)
idf_smell = log10(N/df_smell)
idf_banana = log10(N/df_banana)
idf_lactic = log10(N/df_lactic)
idf_tart = log10(N/df_tart)
```

```
print('the inverse document frequency of word foam   is: ',idf_foam)
print('the inverse document frequency of word smell  is: ',idf_smell)
print('the inverse document frequency of word banana is: ',idf_banana)
print('the inverse document frequency of word lactic is: ',idf_lactic)
print('the inverse document frequency of word tart   is: ',idf_tart)
```

```
def tf(w):
    r = ''.join([c for c in data[0]['review/text'].lower() if not c in punctuation])
    result = r.split().count(w)
    return result
```

```
tf_foam = tf('foam')
tf_smell = tf('smell')
tf_banana = tf('banana')
tf_lactic = tf('lactic')
tf_tart = tf('tart')
tfidf_foam = tf_foam*idf_foam
tfidf_smell = tf_smell*idf_smell
tfidf_banana = tf_banana*idf_banana
tfidf_lactic = tf_lactic*idf_lactic
tfidf_tart = tf_tart*idf_tart
print('tf-idf score of word foam in the first review is   :',tfidf_foam)
print('tf-idf score of word smell in the first review is  :',tfidf_smell)
print('tf-idf score of word banana in the first review is :',tfidf_banana)
print('tf-idf score of word lactic in the first review is :',tfidf_lactic)
print('tf-idf score of word tart in the first review is   :',tfidf_tart)
```

```
#problem 6
unigramID = dict(zip(range(len(unigram_1000)),unigram_1000))
```

```
wordcount1 = defaultdict(int)  #word shows in review 1 and corresponding times
r = ''.join([c for c in data[0]['review/text'].lower() if not c in punctuation])
for w in r.split():
    wordcount1[w] += 1
wordcount2 = defaultdict(int)
r = ''.join([c for c in data[1]['review/text'].lower() if not c in punctuation])
for w in r.split():
    wordcount2[w] += 1
```

```
word_in_1 = [w for w in wordcount1]
word_in_2 = [w for w in wordcount2]
```

```
tf1 = []
for i in range(1000):
    word = unigramID[i]
    if word in word_in_1:
        tf1.append(wordcount1[word])
    else:
        tf1.append(0)
tf2 = []
for i in range(1000):
    word = unigramID[i]
    if word in word_in_2:
        tf2.append(wordcount2[word])
    else:
        tf2.append(0)
idf1 = []
for i in range(1000):
    word = unigramID[i]
    if word in word_in_1:
        idf = log10(N/count(word))
        idf1.append(idf)
    else:
        idf1.append(0)
idf2 = []
for i in range(1000):
    word = unigramID[i]
    if word in word_in_2:
        idf = log10(N/count(word))
        idf2.append(idf)
    else:
        idf2.append(0)

import numpy as np
tfidf1 = np.array(tf1)*np.array(idf1)
tfidf2 = np.array(tf2)*np.array(idf2)
tfidf1_list = tfidf1.tolist()
tfidf2_list = tfidf2.tolist()
sum1 = 0
for i in tfidf1_list:
    sum1 += i**2
sum1 = sum1**(0.5)
sum2 = 0
for i in tfidf2_list:
    sum2 += i**2
sum2 = sum2**(0.5)
from math import pi
cos = np.dot(tfidf1_list,tfidf2_list)/(sum1*sum2)
angle = np.arccos(cos)
print('cos_theta = ',cos,' theta = ',angle/pi,'pi')
```

```
cos_theta =  0.106130241679  theta =  0.466153952679 pi
```

```
#problem 7
unigramID = dict(zip(range(len(unigram_1000)),unigram_1000))
word_ID = dict(zip(unigram_1000,range(len(unigram_1000))))
idf_unigram1000 = [0]*1000
for word in unigram_1000:
    index = word_ID[word]
    idf_unigram1000[index] = log10(5000/count(word))  #calculate the idf for 1000 popular unigrams
```

```
def calculate_tfidf(datum):
    wordcount = defaultdict(int)  #word shows in review 1 and corresponding times
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
    for w in r.split():
        wordcount[w] += 1
    tf = [0]*1000
    idf_value = [0]*1000
    for word in wordcount:
        if word in unigram_1000:
            index = word_ID[word]
            tf[index] = wordcount[word]
            idf_value[index] = idf_unigram1000[index]
        else:
            continue
    tfidf = np.array(tf)*np.array(idf_value)
    tfidf_list = tfidf.tolist()
    return tfidf_list
```

```
tfidf = [calculate_tfidf(d) for d in data]
```

```
norm = []
for i in tfidf:
    sum_value = np.dot(i,i)
    sum_value = sum_value**(0.5)
    norm.append(sum_value)
```

```
tfidf_1 = tfidf[0]
cos_similarity = []
for i in range(5000):
    cos = np.dot(tfidf_1,tfidf[i])
    cos = cos/norm[0]
    cos_similarity.append(cos)
for i in range(5000):
    dd = norm[i]
    cos_similarity[i] = cos_similarity[i]/dd
index = [cos_similarity.index(x) for x in sorted(cos_similarity,reverse = True)[:3]]
data[index[1]]
```

```
#problem 8
new_feature = []
for i in range(5000):
    add_c = tfidf[i]
    add_c.append(1)
    new_feature.append(add_c)
```

```
y = [d['review/overall'] for d in data]
clf = linear_model.Ridge(1.0,fit_intercept=False)
clf.fit(new_feature,y)
theta = clf.coef_
predictions = clf.predict(new_feature)
MSE = 0
for i in range(len(y)):
    MSE = (y[i]-predictions[i])**2
MSE = MSE/len(y)
print('MSE for tfidf model is: ',MSE)
```