Homework 4

CSE 258
Winter 2017

Xiaowen Mao
PID: A53220159
x9mao@eng.ucsd.edu

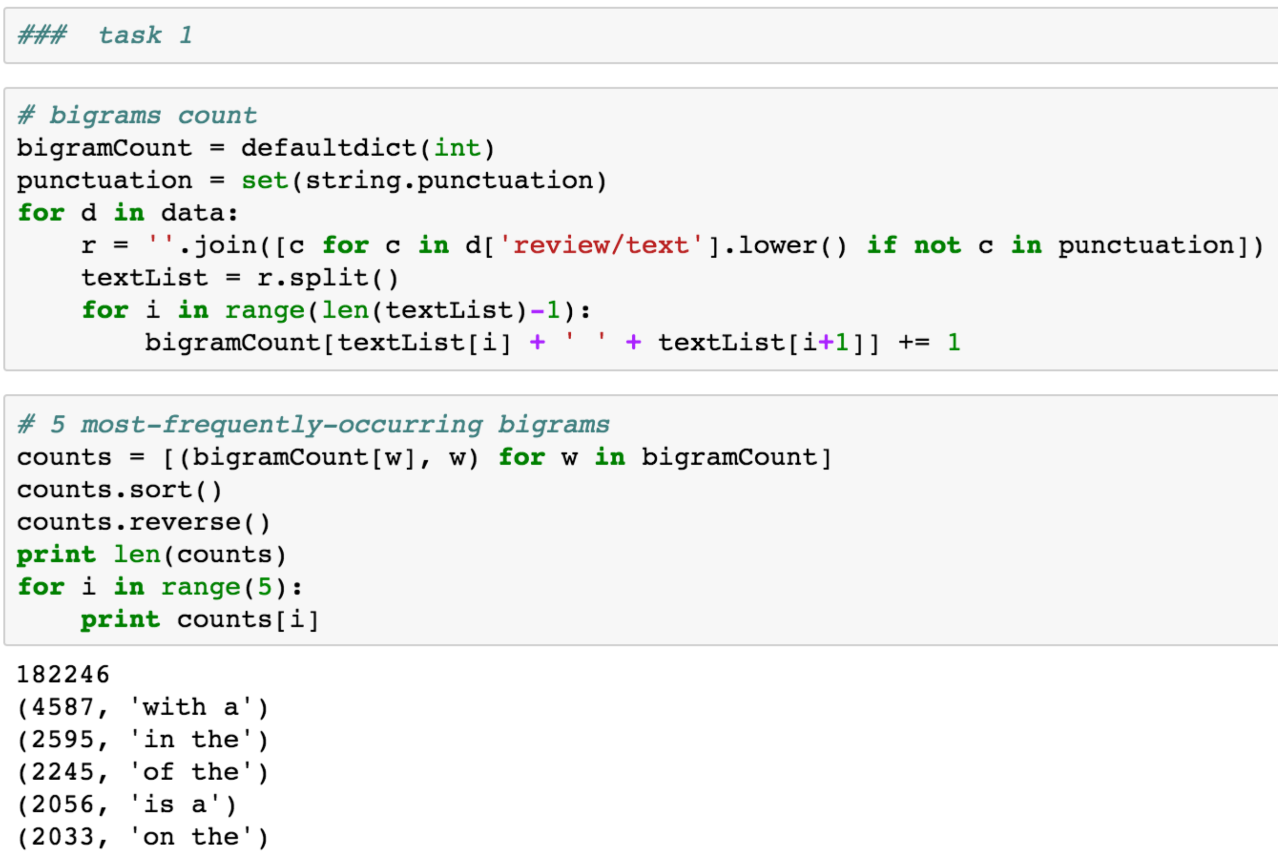
1. Unique Bigrams

1. How many: 182246

2. 5 Most-frequently-occurring:

Bigram	Counts
with a	4587
in the	2595
of the	2245
is a	2056
on the	2033

The source code is as showed in Figure 1.



```
###  task 1
```

```
# bigrams count
bigramCount = 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])
    textList = r.split()
    for i in range(len(textList)-1):
        bigramCount[textList[i] + ' ' + textList[i+1]] += 1
```

```
# 5 most-frequently-occurring bigrams
counts = [(bigramCount[w], w) for w in bigramCount]
counts.sort()
counts.reverse()
print len(counts)
for i in range(5):
    print counts[i]
```

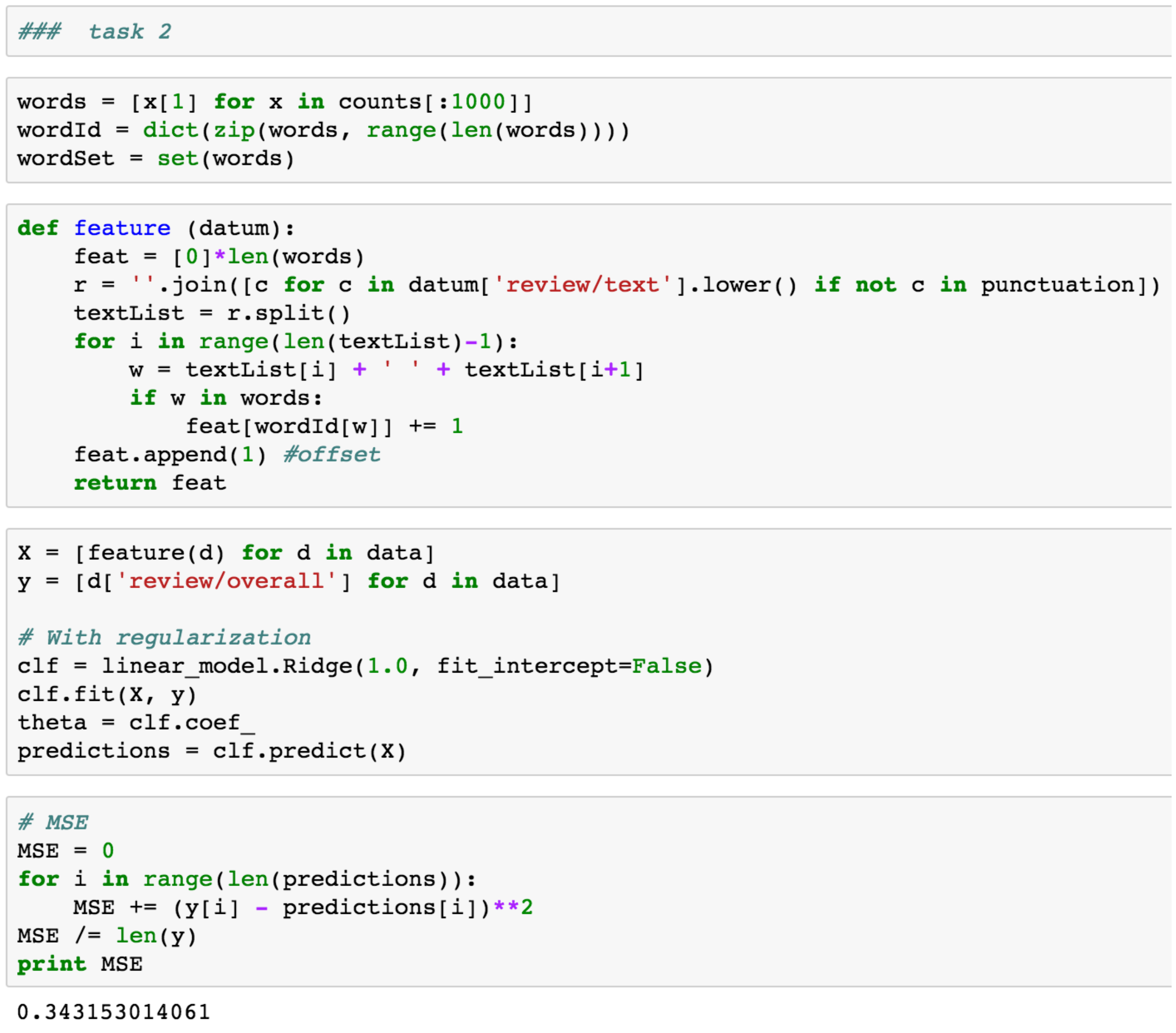
```
182246
(4587, 'with a')
(2595, 'in the')
(2245, 'of the')
(2056, 'is a')
(2033, 'on the')
```

Figure 1: Code for Task 1

2. MSE on 1K Bigrams

$$MSE = 0.343153014061$$

The source code is as showed in Figure 2.



```
###  task 2
```

```
words = [x[1] for x in counts[:1000]]
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])
    textList = r.split()
    for i in range(len(textList)-1):
        w = textList[i] + ' ' + textList[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]

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

```
# MSE
MSE = 0
for i in range(len(predictions)):
    MSE += (y[i] - predictions[i])**2
MSE /= len(y)
print MSE
```

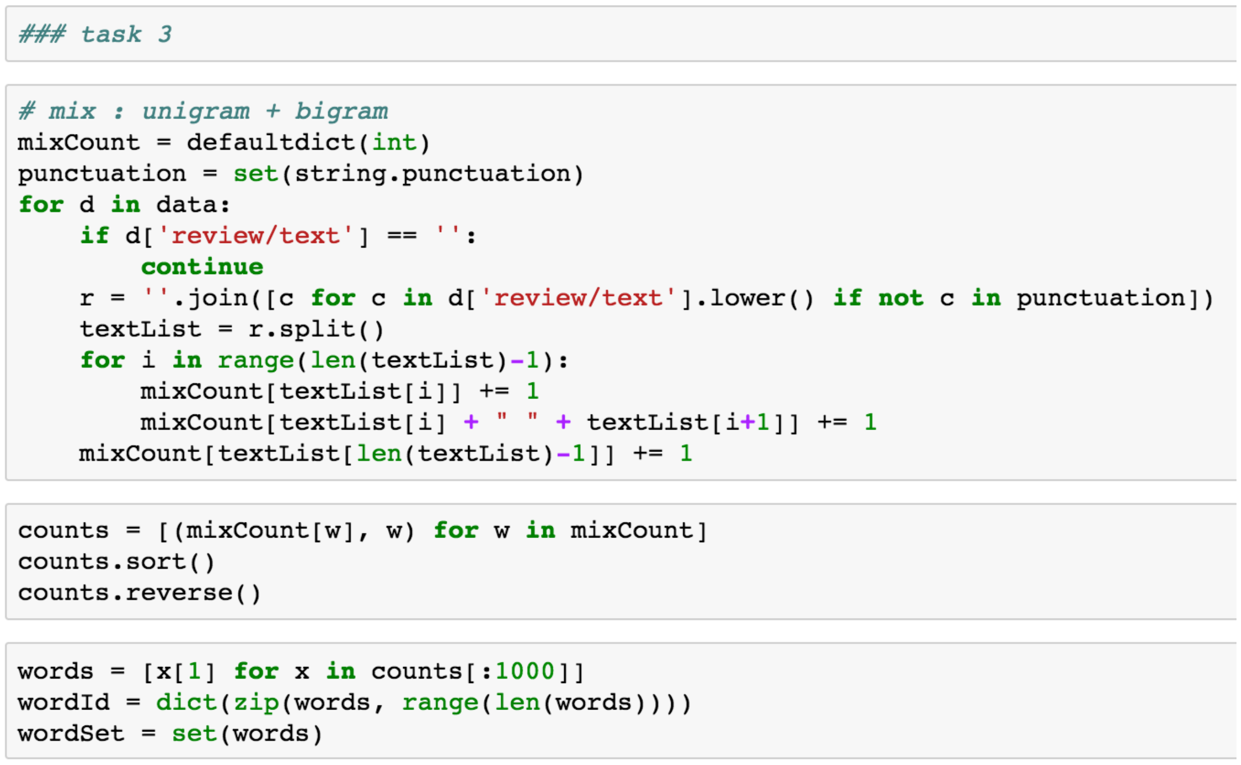
```
0.343153014061
```

Figure 2: Code for Task 2

3. MSE on 1K Combination

$$MSE = 0.289047333034$$

The new predictor is as showed in Figure 3.



```
### task 3
```

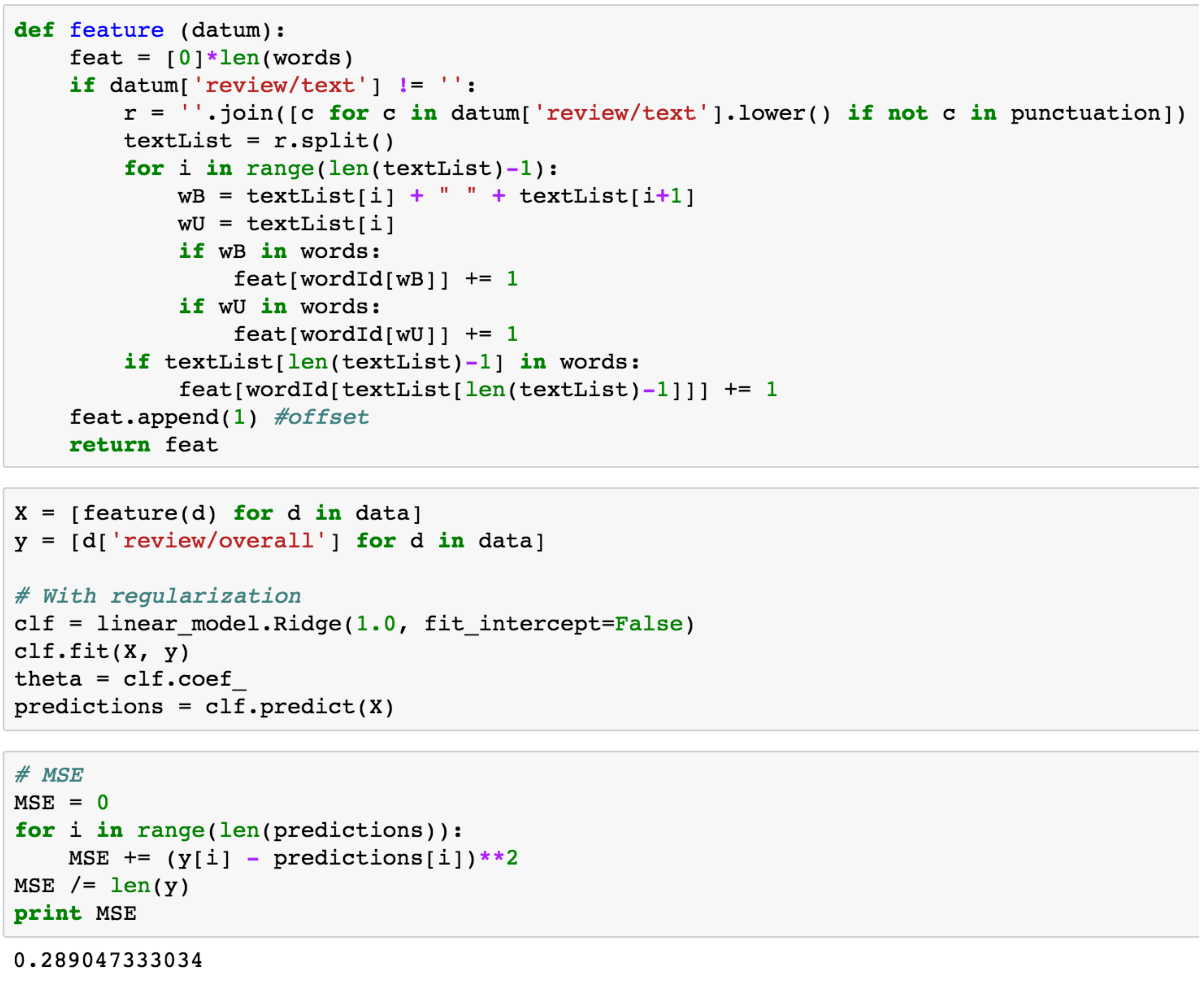
```
# mix : unigram + bigram
mixCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
    if d['review/text'] == '':
        continue
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    textList = r.split()
    for i in range(len(textList)-1):
        mixCount[textList[i]] += 1
        mixCount[textList[i] + " " + textList[i+1]] += 1
    mixCount[textList[len(textList)-1]] += 1
```

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

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

Figure 3: Predictor for Task 3

The result is as showed in Figure 4.



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

```python
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)
```

```python
# MSE
MSE = 0
for i in range(len(predictions)):
    MSE += (y[i] - predictions[i])**2
MSE /= len(y)
print MSE
```

```
0.289047333034
```

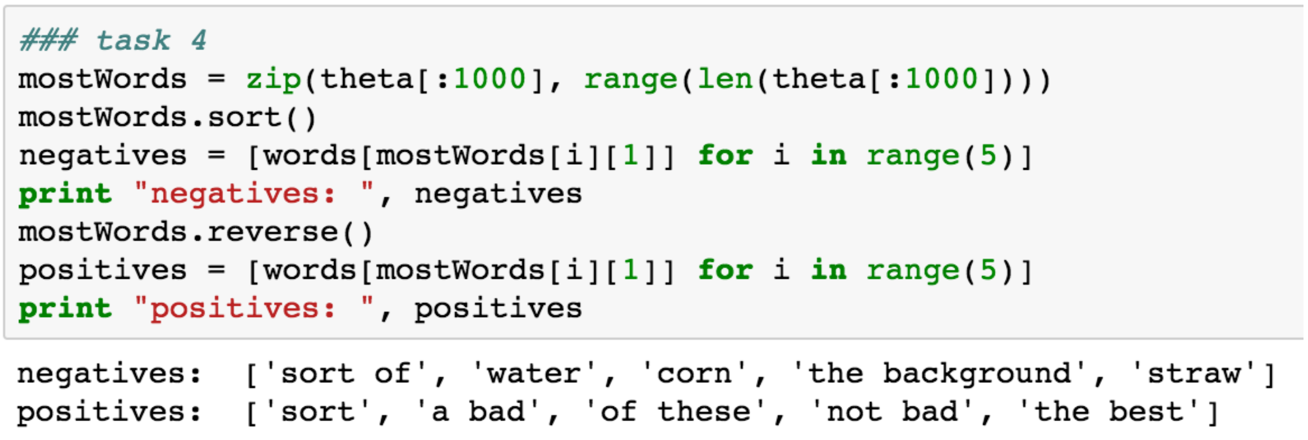
Figure 4: Result for Task 3

4. Most Positive & Negative

Negative: sort of, water, corn, the background, straw

Positive: sort, a bad, of these, not bad, the best

The code and result are attached in Figure 5.



```
### task 4
mostWords = zip(theta[:1000], range(len(theta[:1000])))
mostWords.sort()
negatives = [words[mostWords[i][1]] for i in range(5)]
print "negatives: ", negatives
mostWords.reverse()
positives = [words[mostWords[i][1]] for i in range(5)]
print "positives: ", positives
```

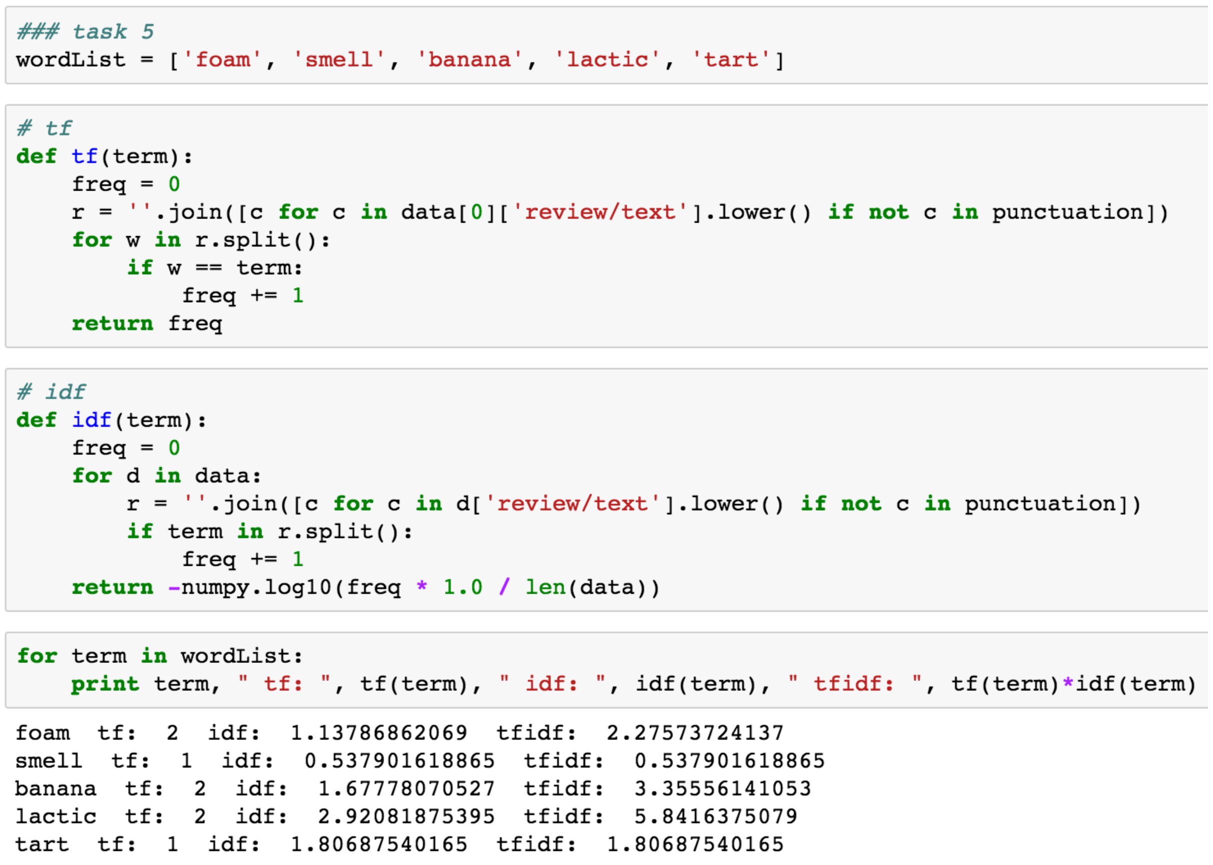
```
negatives:  ['sort of', 'water', 'corn', 'the background', 'straw']
positives:  ['sort', 'a bad', 'of these', 'not bad', 'the best']
```

Figure 5: Code and Result for Task 4

5. IDF & TF-IDF

Term	TF	IDF	TF-IDF
foam	2	1.13786862069	2.27573724137
smell	1	0.537901618865	0.537901618865
banana	2	1.67778070527	3.35556141053
lactic	2	2.92081875395	5.8416375079
tart	1	1.80687540165	1.80687540165

The code and result is attached in Figure 6.



```
### task 5
wordList = ['foam', 'smell', 'banana', 'lactic', 'tart']
```

```
# tf
def tf(term):
    freq = 0
    r = ''.join([c for c in data[0]['review/text'].lower() if not c in punctuation])
    for w in r.split():
        if w == term:
            freq += 1
    return freq
```

```
# idf
def idf(term):
    freq = 0
    for d in data:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
        if term in r.split():
            freq += 1
    return -numpy.log10(freq * 1.0 / len(data))
```

```
for term in wordList:
    print term, " tf: ", tf(term), " idf: ", idf(term), " tfidf: ", tf(term)*idf(term)
```

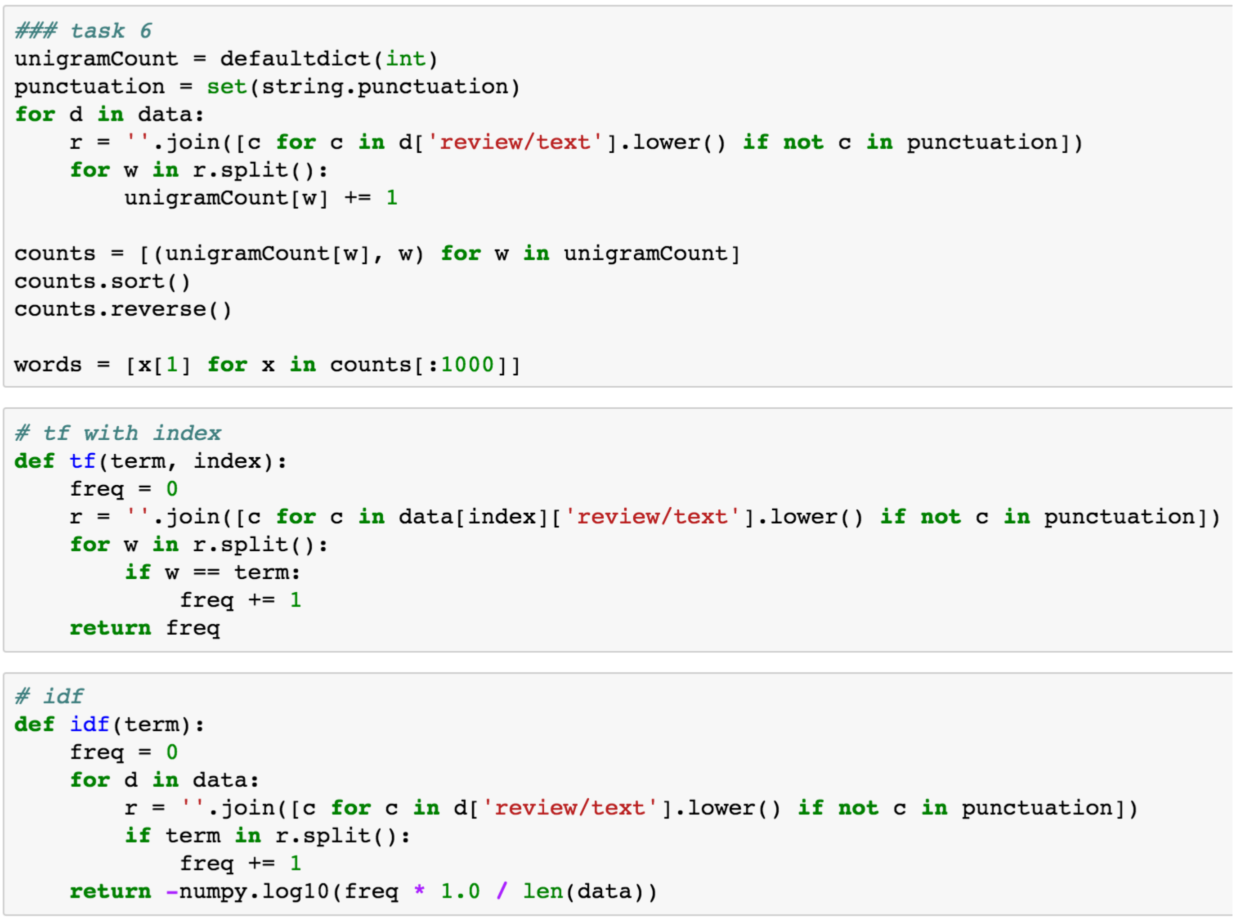
```
foam  tf:  2  idf:  1.13786862069  tfidf:  2.27573724137
smell  tf:  1  idf:  0.537901618865  tfidf:  0.537901618865
banana  tf:  2  idf:  1.67778070527  tfidf:  3.35556141053
lactic  tf:  2  idf:  2.92081875395  tfidf:  5.8416375079
tart  tf:  1  idf:  1.80687540165  tfidf:  1.80687540165
```

Figure 6: Code for Task 5

6. Cosine Similarity

$$CosSim = 0.106130241679$$

The code and result is attached in Figure 7-8.

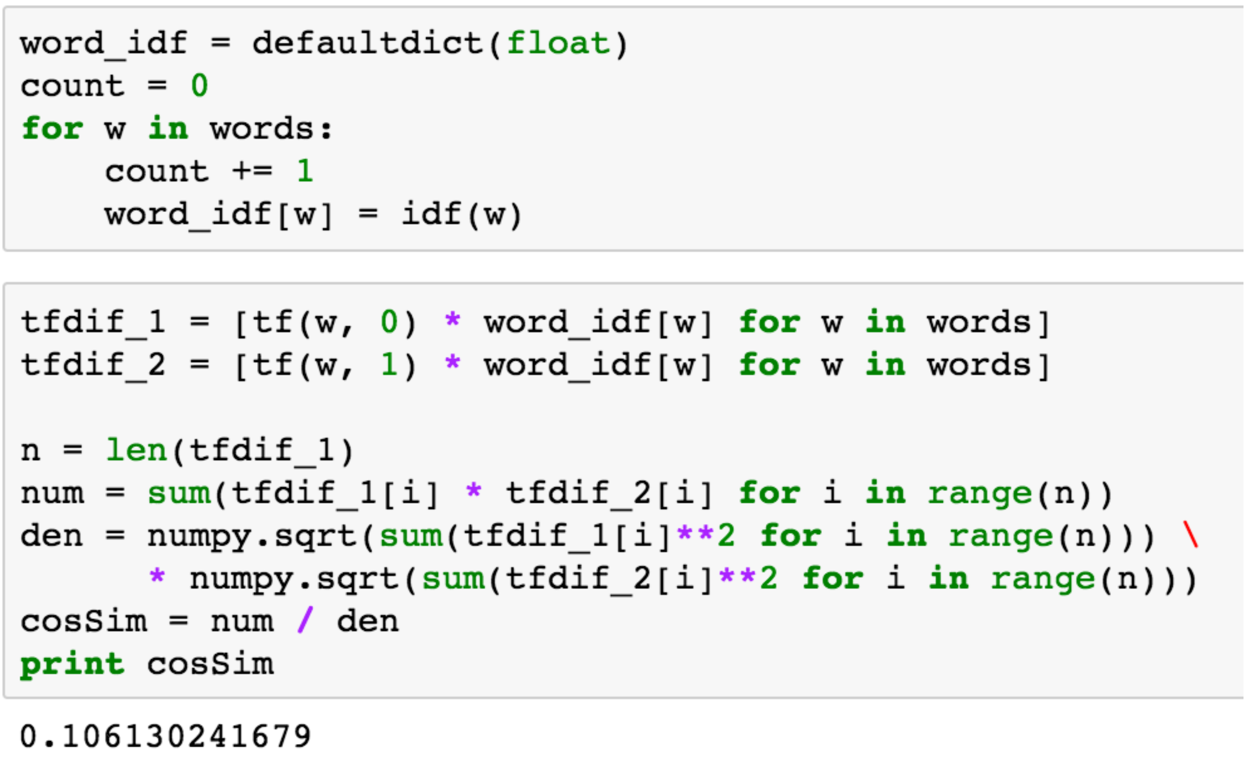


```python
### task 6
unigramCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        unigramCount[w] += 1

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

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

```python
# tf with index
def tf(term, index):
    freq = 0
    r = ''.join([c for c in data[index]['review/text'].lower() if not c in punctuation])
    for w in r.split():
        if w == term:
            freq += 1
    return freq
```

```python
# idf
def idf(term):
    freq = 0
    for d in data:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
        if term in r.split():
            freq += 1
    return -numpy.log10(freq * 1.0 / len(data))
```

Figure 7: Code and result for task 6



```python
word_idf = defaultdict(float)
count = 0
for w in words:
    count += 1
    word_idf[w] = idf(w)
```

```python
tfdif_1 = [tf(w, 0) * word_idf[w] for w in words]
tfdif_2 = [tf(w, 1) * word_idf[w] for w in words]

n = len(tfdif_1)
num = sum(tfdif_1[i] * tfdif_2[i] for i in range(n))
den = numpy.sqrt(sum(tfdif_1[i]**2 for i in range(n))) \
      * numpy.sqrt(sum(tfdif_2[i]**2 for i in range(n)))
cosSim = num / den
print cosSim
```

```
0.106130241679
```

Figure 8: Code and result for task 6

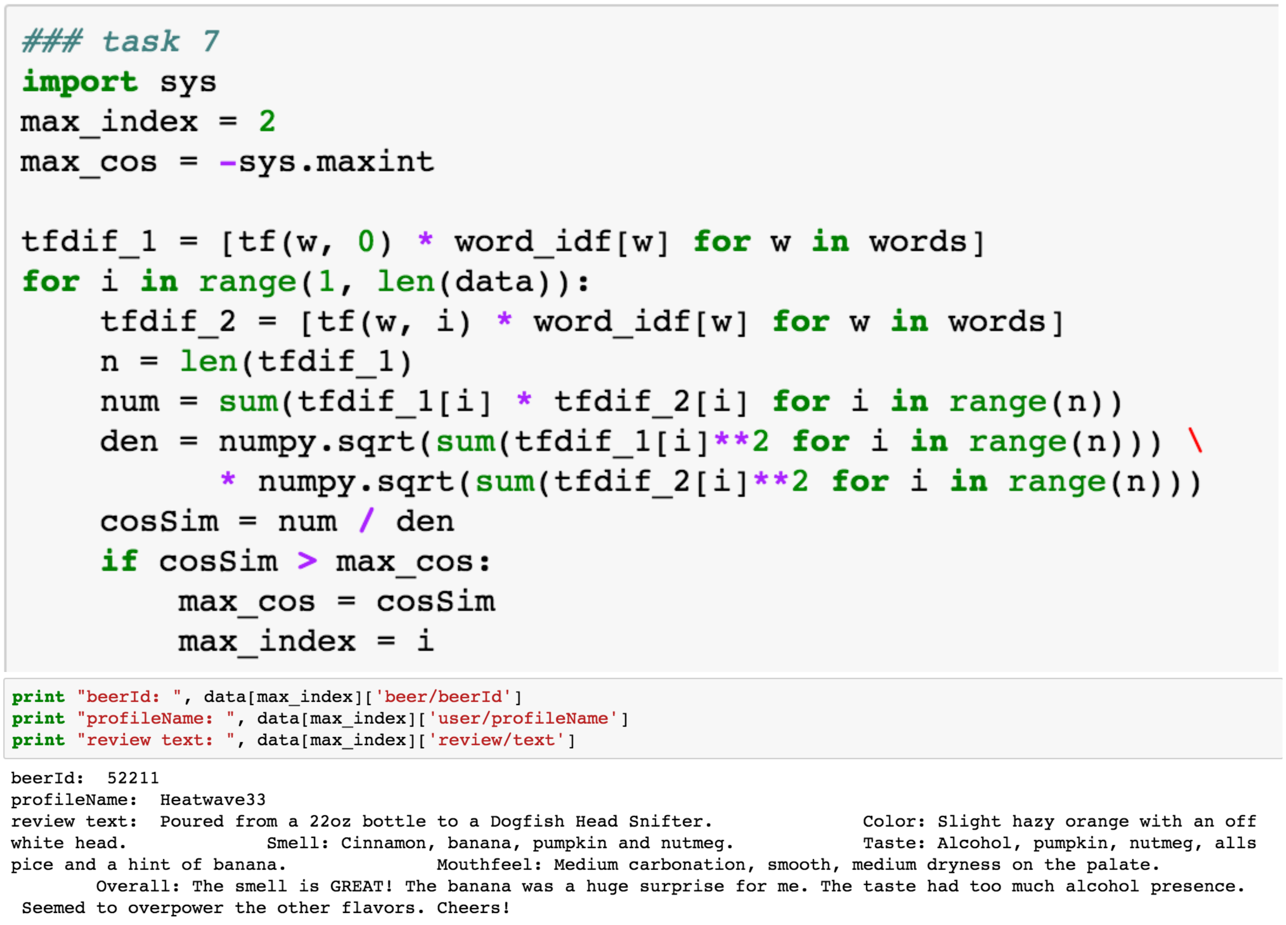
7. Highest Cosine Similarity

beerId: 52211

profileName: Heatwave33

review text: Poured from a 22oz bottle to a Dogfish Head Snifter. Color: Slight hazy orange with an off white head. Smell: Cinnamon, banana, pumpkin and nutmeg. Taste: Alcohol, pumpkin, nutmeg, allspice and a hint of banana. Mouthfeel: Medium carbonation, smooth, medium dryness on the palate. Overall: The smell is GREAT! The banana was a huge surprise for me. The taste had too much alcohol presence. Seemed to overpower the other flavors. Cheers!

The code and result is attached in Figure 9.



```
### task 7
import sys
max_index = 2
max_cos = -sys.maxint

tfdif_1 = [tf(w, 0) * word_idf[w] for w in words]
for i in range(1, len(data)):
    tfdif_2 = [tf(w, i) * word_idf[w] for w in words]
    n = len(tfdif_1)
    num = sum(tfdif_1[i] * tfdif_2[i] for i in range(n))
    den = numpy.sqrt(sum(tfdif_1[i]**2 for i in range(n))) \
          * numpy.sqrt(sum(tfdif_2[i]**2 for i in range(n)))
    cosSim = num / den
    if cosSim > max_cos:
        max_cos = cosSim
        max_index = i
```

```
print "beerId: ", data[max_index]['beer/beerId']
print "profileName: ", data[max_index]['user/profileName']
print "review text: ", data[max_index]['review/text']
```

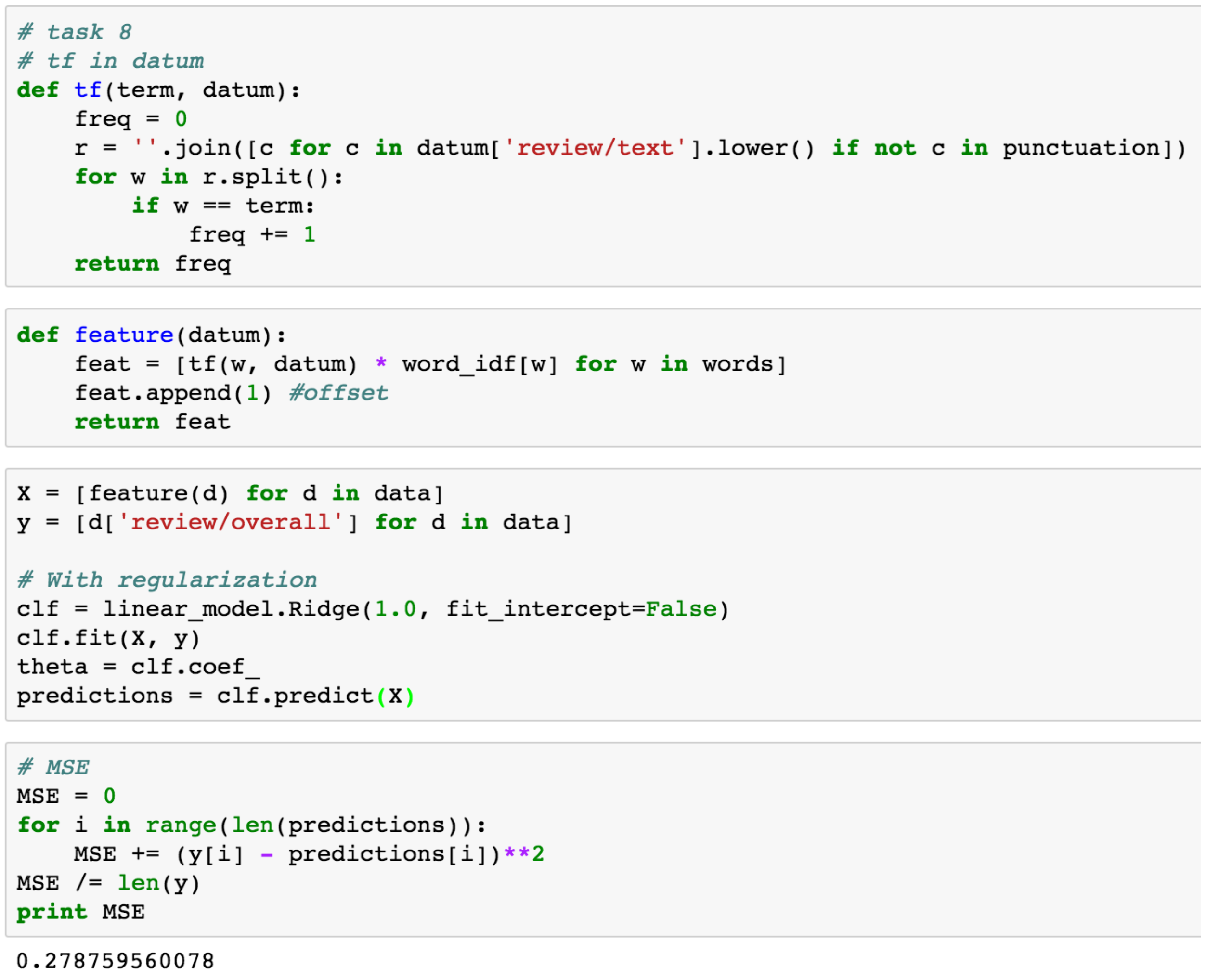
```
beerId:  52211
profileName:  Heatwave33
review text:  Poured from a 22oz bottle to a Dogfish Head Snifter.              Color: Slight hazy orange with an off
white head.             Smell: Cinnamon, banana, pumpkin and nutmeg.            Taste: Alcohol, pumpkin, nutmeg, alls
pice and a hint of banana.              Mouthfeel: Medium carbonation, smooth, medium dryness on the palate.
        Overall: The smell is GREAT! The banana was a huge surprise for me. The taste had too much alcohol presence.
 Seemed to overpower the other flavors. Cheers!
```

Figure 9: Code and result for task 7

8. 1000-Dimensional TF-IDF

$$MSE = 0.278759560078$$

The code and result is attached in Figure 8.



```
# task 8
# tf in datum
def tf(term, datum):
    freq = 0
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
    for w in r.split():
        if w == term:
            freq += 1
    return freq
```

```
def feature(datum):
    feat = [tf(w, datum) * word_idf[w] for w in words]
    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)
```

```
# MSE
MSE = 0
for i in range(len(predictions)):
    MSE += (y[i] - predictions[i])**2
MSE /= len(y)
print MSE
```

```
0.278759560078
```

Figure 10: Code for task 8