I. Description of the dataset

The number of fields is 2, they are: business_name and review_text.
The size of the dataset is: 14220 review text.
The average number of sentences for each review text and the average number of words for each review text are as following:

```
The average number of sentences for each review is: 12.048312236286920
The average number of words for each review is: 209.40007032348805
```

II. The sentiment of the review comments at the sentence level

1. Processing techniques for data preprocessing task

For processing the review text, First I will split the review text into sentences.

```
# split the text into sentences
def split_sentences(content):
    sentences = nltk.sent_tokenize(content)
    return sentences
```

I remove the “\r\n”, “\r”, “\n”, “@” which are not useful for following analysis. And I do the word tokenize for the review text.

```
# do tokenization
def do_word_tokenize(content):
    # do word tokenizing process
    content = content.replace("\r\n", " ")
    content = content.replace("\r", " ")
    content = content.replace("\n", " ")
    content = content.replace("@", " ")
    tokens = nltk.word_tokenize(content)
    return tokens
```

Second I lower the words in the review text to avoid the capital letter to impact the following analysis.

```
# do lower the words
def do_lower(content):
    # set all words as lowercase
    words = [w.lower() for w in content]
    return words
```

2. Features I used for the classification task

I use sentence_polarity as the corpus for the classification.

First I get the sentence, category pair from sentence_polarity. Then use random to mix them up for later separation into training and test sets.

```
# get the sentence, category pairs
def get_document():
    documents = [(sent, category) for category in
sentence_polarity.categories() for sent in
                sentence_polarity.sents(categories=category)]
    random.shuffle(documents)
    return documents
```

Second, I get the word features from the sentence, category pairs.

I get set of words for features which do not do preprocessing for the data.

Get all words list and then get the top 2000 words from the words list.

```
# get set of words for features
def get_word_features(documents):
    all_words_list = [word for (sent, category) in documents for word in
sent]
    all_words = nltk.FreqDist(all_words_list)
    word_items = all_words.most_common(2000)
    word_features = [word for (word, freq) in word_items]
    return word_features
```

Then I also get set of words for features which remove the stop words and reserve the words list which have meanings about sentiment.

```
# filter stop words's words features
def filter_stop_words(documents):
    all_words_list = [word for (sent, category) in documents for word in
sent]
    stopwords = nltk.corpus.stopwords.words('english')
    newstopwords = [word for word in stopwords if word not in new_list]
    new_all_words_list = [word for word in all_words_list if word not in
newstopwords]
    new_all_words = nltk.FreqDist(new_all_words_list)
    new_word_items = new_all_words.most_common(2000)
    new_word_features = [word for (word, count) in new_word_items]
    return new_word_features
```

The words list which may have meanings about sentiment:

```
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing',
'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither',
'nor']
reservewords = ['again', 'more', 'most', 'only', 'too', 'very']
notmeaningwords = ["don", "don't",'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
new_list = negationwords + reservewords + notmeaningwords
```

Then I have three kinds of feature: unigram features, subjectivity count feature, negation feature.

2.1. Unigram feature:

For unigram features, I will get the unigram feature whose label will be 'contains(keyword)' for each keyword in the word_features set, and the value of the feature will be Boolean which means whether the word is contained in the document.

```
# get unigram feature
def document_features_contain(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

Then I will use the unigram features to train the classifier. The corresponding accuracy will also be printed out.

```
# use the unigram feature to train the classifier
def document_features_contain_train(documents, word_features):
    featuresets = [(document_features_contain(d, word_features), c) for (d,
c) in documents]
    train_set, test_set = featuresets[1000:], featuresets[:1000]
    classifier = nltk.NaiveBayesClassifier.train(train_set)
    print (nltk.classify.accuracy(classifier, test_set))
    return test_set, classifier
```

Then this classifier's accuracy, precision, recall and F-measure are as following:

```
0.74
For pos
pos precision: 0.7349896480331263
pos recall: 0.728952772073922
pos F-measure: 0.731958762886598
For neg
neg precision: 0.7446808510638298
neg recall: 0.7504873294346979
neg F-measure: 0.7475728155339807
```

2.2. Subjectivity count feature:

For subjectivity count feature, I will create a subjectivity lexicon represented as dictionary.

```
# creates a Subjectivity Lexicon
def readSubjectivity(path):
    flexicon = open(path, 'r')
    # initialize an empty dictionary
    sldict = {}
    for line in flexicon:
        fields = line.split()   # default is to split on whitespace
        # split each field on the '=' and keep the second part as the value
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        # put a dictionary entry with the word as the keyword
        #     and a list of the other values
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict
```

Then I have two kinds of feature extraction function, First one will count all the positive and negative subjectivity words. And weakly subjective words will be counted once and strongly subjective words will be counted twice.

```
# get SL features
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
```

Then I will use the subjectivity lexicon feature to train the naïve bayes classifier.

```
# use SL features to train
def SL_features_train(documents, word_features, SL):
    SL_featuresets = [(SL_features(d, word_features, SL), c) for (d, c) in
documents]
    train_set, test_set = SL_featuresets[1000:], SL_featuresets[:1000]
    classifier = nltk.NaiveBayesClassifier.train(train_set)
    print(nltk.classify.accuracy(classifier, test_set))
    return test_set, classifier
```

Then this classifier's accuracy, precision, recall and F-measure are as following:

```
0.761
For pos
pos precision: 0.7533206831119544
pos recall: 0.7845849802371542
pos F-measure: 0.7686350435624395
For neg
neg precision: 0.7695560253699789
neg recall: 0.7368421052631579
neg F-measure: 0.7528438469493278
```

Second one will count all the positive and negative subjectivity words. And weak positive, strong positive, weak negative, strong negative will be counted respectively.

```
# get SL features
def new_SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['wpositivecount'] = weakPos
            features['sposativecount'] = strongPos
            features['wnegitivecount'] = weakNeg
            features['snegativecount'] = strongNeg
    return features
```

Then I will use the subjectivity lexicon feature to train the naïve bayes classifier.

```
# use SL features to train
def new_SL_features_train(documents, word_features, SL):
    SL_featuresets = [(new_SL_features(d, word_features, SL), c) for (d, c)
in documents]
    train_set, test_set = SL_featuresets[1000:], SL_featuresets[:1000]
    classifier = nltk.NaiveBayesClassifier.train(train_set)
    print(nltk.classify.accuracy(classifier, test_set))
    return test_set, classifier
```

Then this classifier's accuracy, precision, recall and F-measure are as following:

```
0.772
For pos
pos precision: 0.7854166666666667
pos recall: 0.750996015936255
pos F-measure: 0.7678207739307537
For neg
neg precision: 0.7596153846153846
neg recall: 0.7931726907630522
neg F-measure: 0.7760314341846757
```

2.3. Negation feature

For negation feature, If a negation occurs and it is in the top 2000 feature words, add the following word as a Not word feature, and otherwise add it as a regular feature word. Then we can get negation feature.

```
# negation features
def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = False
        features['contains(NOT{})'.format(word)] = False  # go through
document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or
(word.endswith("n't"))):
            i += 1
            features['contains(NOT{})'.format(document[i])] = (document[i] in
word_features)
        else:
            features['contains({})'.format(word)] = (word in word_features)
    return features
```

Then use the negation feature to train the naïve bayes classifier.

```
# use negation features to train
def NOT_features_train(documents, word_features, negationwords):
   NOT_featuresets = [(NOT_features(d, word_features, negationwords), c) for
(d, c) in documents]
   train_set, test_set = NOT_featuresets[200:], NOT_featuresets[:200]
   classifier = nltk.NaiveBayesClassifier.train(train_set)
   print(nltk.classify.accuracy(classifier, test_set))
   return test_set, classifier
```

Then this classifier's accuracy, precision, recall and F-measure are as following:

```
0.765
For pos
pos precision: 0.7425742574257426
pos recall: 0.78125
pos F-measure: 0.7614213197969543
For neg
neg precision: 0.7878787878787878
neg recall: 0.75
neg F-measure: 0.7684729064039408
```

2.4. Selection of the feature

For here, I will use second kinds of Subjectivity count feature since this one's classifier measure is a little better than others. This kind of Subjectivity count feature will count weak positive, strong positive, weak negative, strong negative respectively.

2.5. Filter out non-alpha characters in the feature set

For here, I want to filter out the non-alpha characters for the set of words for features. However, when I filter out the non-alpha characters, and use my selected feature to train the naïve bayes classifier, the measurement of the classifier is as following:

```
0.761
For pos
pos precision: 0.7566037735849057
pos recall: 0.7847358121330724
pos F-measure: 0.7704130643611912
For neg
neg precision: 0.7659574468085106
neg recall: 0.7361963190184049
neg F-measure: 0.7507820646506779
```

However, the measure scores become lower. For here I will not filter out the non-alpha characters. And for the reason why this decrease the measure scores, I think some

special characters may also have sentiment meanings such as '!', '?'.

3. Classify the review data by using the trained classifier

I will get the features for each sentence of review text and then use trained classifier to classify each sentence. And then I will get two lists of the positive sentences and negative sentences. At last, I will store the two list sentences into two files which are pos_sentences.txt and neg_sentences.txt.

```
pos_sentences = []
neg_sentences = []
csv_data = pd.read_csv('13501-27721 upload_revised.csv')  # use panda to
read the csv file
for num in range(0, 14220):
    review_text_data = csv_data.loc[num, 'review_text']
    business_name_data = csv_data.loc[num, 'business_name']
    sentences = split_sentences(review_text_data)  # split text into
sentences
    for sentence in sentences:
        filter_words = preprocess_data(sentence)  # preprocess the data
        inputfeatureset = document_features_contain(filter_words,
word_features)  # get review data features
        if(classifier.classify(inputfeatureset) == 'pos'):
            pos_sentences.append(sentence)
        else:
            neg_sentences.append(sentence)
print(pos_sentences)
print(neg_sentences)

file_pos = open('pos_sentences.txt', mode='w')  # store positive sentences
file_neg = open('neg_sentences.txt', mode='w')  # store negative sentences
for pos_sentence in pos_sentences:
    pos_sentence = pos_sentence.replace("\r\n", " ")
    pos_sentence = pos_sentence.replace("\r", " ")
    pos_sentence = pos_sentence.replace("\n", " ")
    if(pos_sentence == " "):
        continue
    file_pos.write(pos_sentence)
    file_pos.write('\n')
file_pos.close()
for neg_sentence in neg_sentences:
    neg_sentence = neg_sentence.replace("\r\n", " ")
    neg_sentence = neg_sentence.replace("\r", " ")
    neg_sentence = neg_sentence.replace("\n", " ")
    if(neg_sentence == " "):
```

```
        continue
    file_neg.write(neg_sentence)
    file_neg.write('\n')
file_neg.close()
```

III. Pattern observed

When I do the classification, there are some most informative features which are top ranked features. I will show the top 30 informative features.

```
Most Informative Features
    contains(engrossing) = True              pos : neg    =     17.6 : 1.0
       contains(generic) = True              neg : pos    =     17.0 : 1.0
      contains(mediocre) = True              neg : pos    =     16.4 : 1.0
      contains(captures) = True              pos : neg    =     16.3 : 1.0
         contains(waste) = True              neg : pos    =     15.7 : 1.0
       contains(routine) = True              neg : pos    =     14.4 : 1.0
    contains(refreshing) = True              pos : neg    =     13.6 : 1.0
          contains(flat) = True              neg : pos    =     13.0 : 1.0
          contains(dull) = True              neg : pos    =     12.5 : 1.0
          contains(warm) = True              pos : neg    =     12.2 : 1.0
     contains(wonderful) = True              pos : neg    =     11.8 : 1.0
    contains(thoughtful) = True              pos : neg    =     11.8 : 1.0
  contains(refreshingly) = True              pos : neg    =     11.6 : 1.0
      contains(touching) = True              pos : neg    =     11.5 : 1.0
 contains(extraordinary) = True              pos : neg    =     11.0 : 1.0
         contains(vivid) = True              pos : neg    =     11.0 : 1.0
        contains(beauty) = True              pos : neg    =     10.6 : 1.0
      contains(provides) = True              pos : neg    =     10.6 : 1.0
         contains(stale) = True              neg : pos    =     10.4 : 1.0
        contains(stupid) = True              neg : pos    =     10.2 : 1.0
      contains(mindless) = True              neg : pos    =      9.7 : 1.0
    contains(meandering) = True              neg : pos    =      9.7 : 1.0
   contains(mesmerizing) = True              pos : neg    =      9.6 : 1.0
      contains(poignant) = True              pos : neg    =      9.4 : 1.0
          contains(loud) = True              neg : pos    =      9.0 : 1.0
      contains(supposed) = True              neg : pos    =      9.0 : 1.0
        contains(unless) = True              neg : pos    =      9.0 : 1.0
      contains(annoying) = True              neg : pos    =      9.0 : 1.0
       contains(harvard) = True              neg : pos    =      9.0 : 1.0
      contains(tiresome) = True              neg : pos    =      9.0 : 1.0
```

These features are most informative features and influence the following sentence's classification. From above statistics, we can get that some words such as mediocre, waste, dull tend to be negative. And some words such as wonderful, thoughtful, beauty tend to be positive.

Then I analyze the two list of sentences which are the sentiment analysis results.
For positive sentences, I analyze the top 50 frequent words. For each sentence, I will do some preprocesses: get content in the pos_sentences.txt, do word tokenize, do lemmatization, lower the words, get alphabetical words, filter stop words. Then I get the top 50 frequent words as following:

```
("'s", 6302)
('food', 5614)
('great', 5052)
("n't", 4643)
('not', 4399)
('place', 4202)
('good', 4020)
('very', 3960)
('get', 3315)
('service', 3101)
('also', 3008)
('one', 2886)
('time', 2704)
('love', 2698)
('make', 2677)
('come', 2529)
('go', 2498)
('nice', 2494)
('well', 2380)
('like', 2292)
('order', 2160)
('best', 2139)
('u', 2053)
('take', 1895)
('always', 1865)
('back', 1788)
("'ve", 1762)
('look', 1759)
('give', 1750)
('first', 1750)
('experience', 1728)
('work', 1643)
('definitely', 1590)
('try', 1516)
```

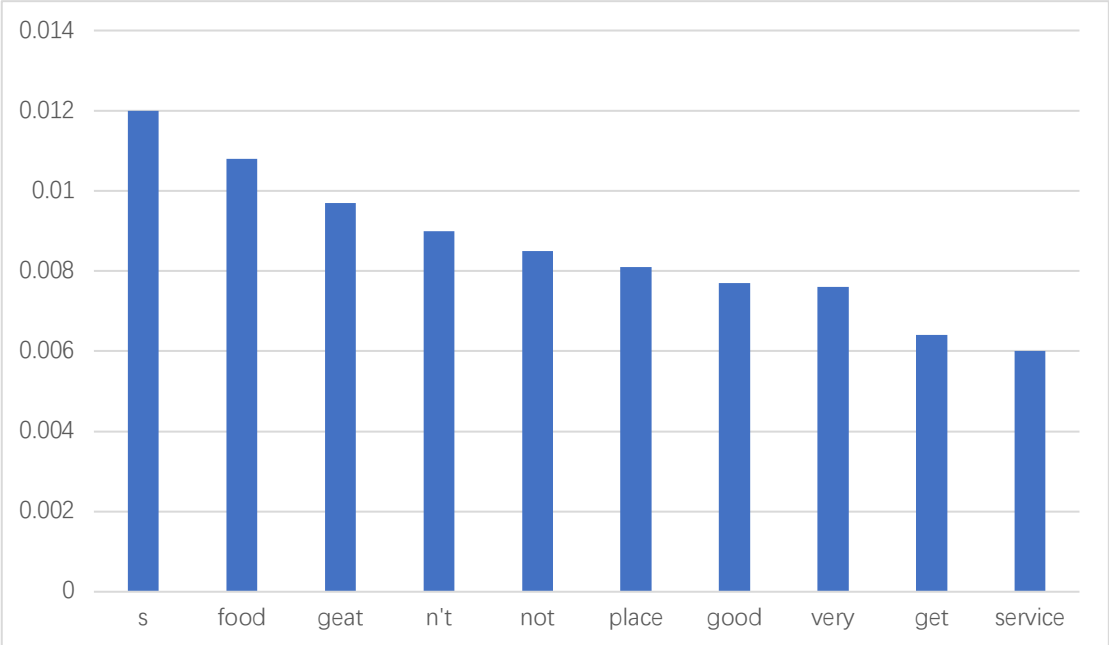
```
('more', 1483)
('see', 1470)
('find', 1466)
('restaurant', 1440)
('know', 1404)
("'m", 1387)
('still', 1365)
('friendly', 1328)
('fresh', 1323)
('really', 1308)
('enjoy', 1305)
('even', 1287)
('year', 1282)
('price', 1276)
('two', 1250)
('right', 1230)
```

From above top 50 frequent words in positive sentences, we can get to know that there are many positive words: great, good, love, nice, friendly, enjoy. These words appear frequently and can lead to positive sentiment. And words like foods, restaurant appear frequently because I analyze the Yelp review data.
And I also get the proportion of each top 50 frequent words.

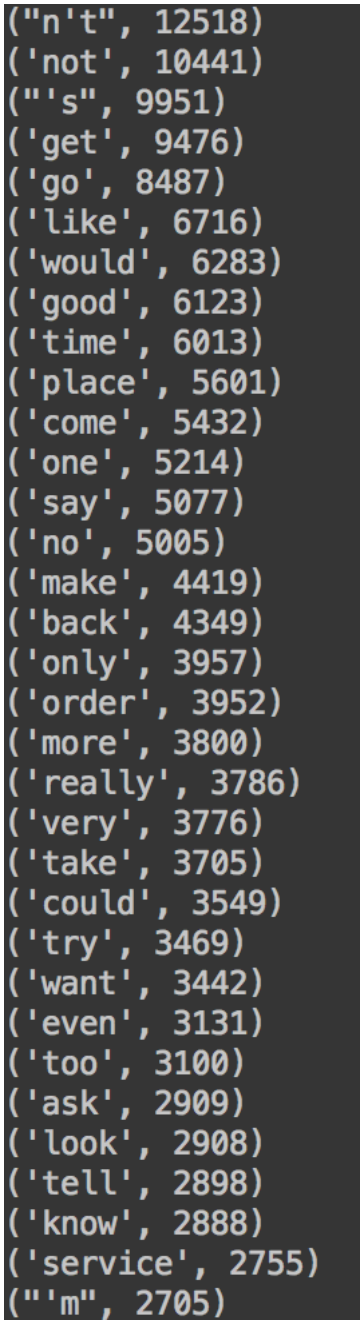
```
's 0.012223317014888319
food 0.010888876820308318
great 0.009798825382293841
n't 0.009005531720108928
not 0.008532270953426485
place 0.008150171072129595
good 0.00779716509042384
very 0.007680789492059305
get 0.006429751809640554
service 0.00601467884214038
also 0.0058342966646753504
one 0.00559766628133413
time 0.005244660299628374
love 0.005233022739791921
make 0.005192291280364333
come 0.004905231471065147
go 0.004845104078576804
nice 0.004837345705352502
well 0.004616232068459885
like 0.004445547857525234
order 0.004189521541123257
best 0.004148790081695 6705
u 0.00398198505737317
take 0.003675529315013228
always 0.0036173415158309605
back 0.003467992831263141
've 0.0034175634053051755
look 0.003411744625386949
give 0.003394288285632269
first 0.003394288285632269
experience 0.0033516172328986057
work 0.003186751801882181
definitely 0.0030839533566601753
try 0.0029404234520105823
more 0.002876416872910088
see 0.002851202159931106
find 0.002843443786706803
```

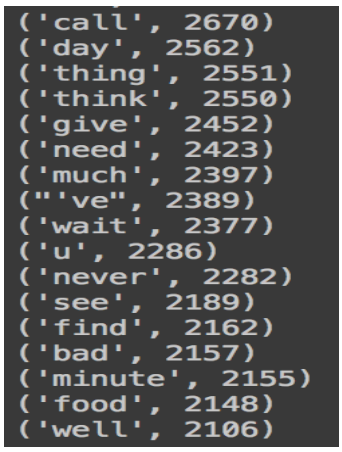
```
restaurant 0.0027930143607488383
know 0.0027231890017301173
'm 0.0026902159155268324
still 0.0026475448627931693
friendly 0.002575779910468373
fresh 0.002566081943937995
really 0.0025369880443468613
enjoy 0.0025311692644286346
even 0.002496256584919274
year 0.0024865586183888963
price 0.0024749210585524426
two 0.0024244916325944776
right 0.002385699766472966
```

Then I get a graph to represent the proportions (top 10 words):



For negative sentences, I analyze the top 50 frequent words. For each sentence, I will do some preprocesses: get content in the neg_sentences.txt, do word tokenize, do lemmatization, lower the words, get alphabetical words, filter stop words. Then I get the top 50 frequent words as following:





From above top 50 frequent words in negative sentences, we can get to know that there

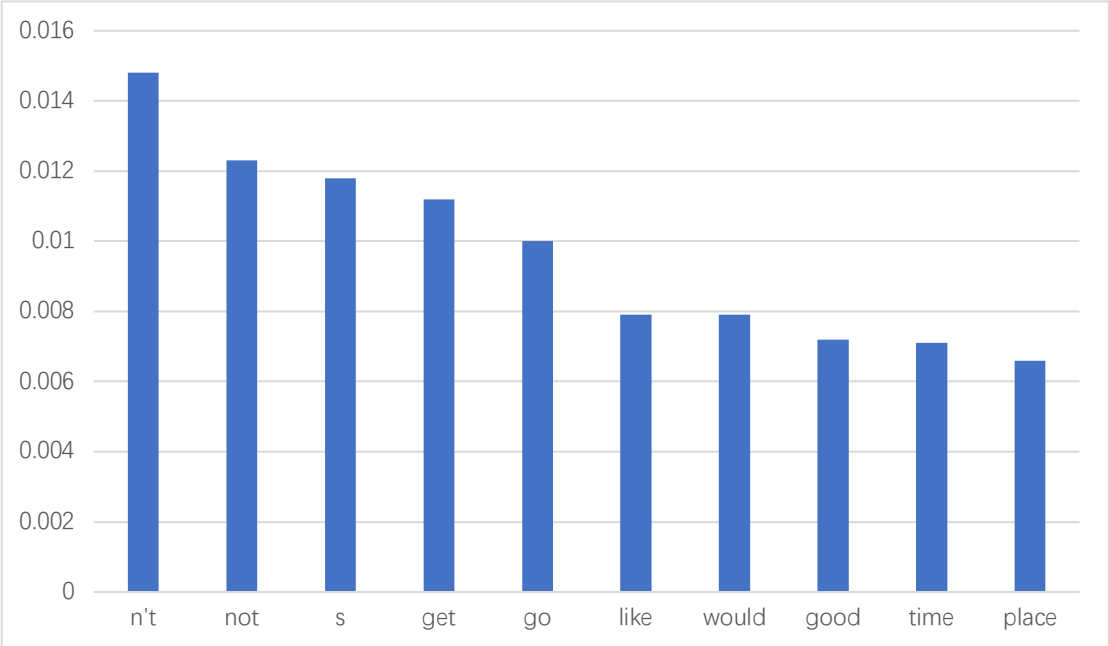
are many negative words: n’t, not, no, never, bad. These words appear frequently and can lead to negative words. What is more, some words like really, very, even, too, much, these words can also strengthen mood and many negative reviews may use this to express the negative sentiment.

And I also get the proportion of each top 50 frequent words.

```
n't 0.014857684086088636
not 0.012392481190513775
's 0.011810897454918359
get 0.011247117303065659
go 0.010073267681629194
like 0.007971257894405758
would 0.0074573277770326645
good 0.0072674228837770185
time 0.007136863269663761
place 0.006647858169530472
come 0.006447271126029196
one 0.006188525708968377
say 0.00602591964411823
no 0.00594046244215319
make 0.005244935770604384
back 0.005161852379805039
only 0.004696585391328705
order 0.004690650863414466
more 0.004510241214821602
really 0.004493624536661733
very 0.004481755480833256
take 0.0043974851844510625
could 0.004212327913526807
try 0.004117375466898983
want 0.004085329016162093
even 0.0037162013798964304
too 0.0036794073068281492
ask 0.0034527083405042213
look 0.0034515214349213736
tell 0.0034396523790928956
know 0.0034277833232644176
service 0.0032699248807456617
'm 0.0032105796016032723
call 0.0031690379062035996
day 0.003040852103256038
```

```
thing 0.0030277961418447123
think 0.0030266092362618647
give 0.0029102924891427 81
need 0.0028758722272401 95
much 0.0028450126820861 53
've 0.0028355174374233706
wait 0.0028212745704291 97
u 0.0027132661623900482
never 0.0027085185400586567
see 0.0025981363208538123
find 0.0025660898701169222
bad 0.002560155342202683
minute 0.0025577815310369877
food 0.002549473191957053
well 0.002499623157477446
```

Then I get a graph to represent the proportions (top 10 words):



IV. Table including two lists of sentences

For the two list of sentences including positive and negative, I will store them into two txt files: pos_sentences.txt and neg_sentences.txt.
CIS668-HW3/neg_sentences.txt
CIS668-HW3/pos_sentences.txt

V. Python code and processing screenshots

Processing screenshots is as following:

```
0.764
Most Informative Features
          contains(warm) = True              pos : neg    =     20.0 : 1.0
    contains(engrossing) = True              pos : neg    =     19.4 : 1.0
      contains(mediocre) = True              neg : pos    =     16.6 : 1.0
      contains(touching) = True              pos : neg    =     16.0 : 1.0
       contains(generic) = True              neg : pos    =     15.9 : 1.0
     contains(inventive) = True              pos : neg    =     15.4 : 1.0
          contains(flat) = True              neg : pos    =     14.4 : 1.0
            contains(90) = True              neg : pos    =     13.2 : 1.0
        contains(boring) = True              neg : pos    =     12.9 : 1.0
    contains(refreshing) = True              pos : neg    =     12.8 : 1.0
  contains(refreshingly) = True              pos : neg    =     12.8 : 1.0
      contains(powerful) = True              pos : neg    =     12.0 : 1.0
     contains(wonderful) = True              pos : neg    =     10.8 : 1.0
     contains(realistic) = True              pos : neg    =     10.8 : 1.0
      contains(provides) = True              pos : neg    =     10.8 : 1.0
         contains(vivid) = True              pos : neg    =     10.8 : 1.0
          contains(dull) = True              neg : pos    =     10.5 : 1.0
       contains(purpose) = True              neg : pos    =     10.5 : 1.0
         contains(stale) = True              neg : pos    =     10.5 : 1.0
        contains(unless) = True              neg : pos    =     10.5 : 1.0
   contains(mesmerizing) = True              pos : neg    =     10.2 : 1.0
        contains(stupid) = True              neg : pos    =      9.9 : 1.0
    contains(apparently) = True              neg : pos    =      9.8 : 1.0
      contains(mindless) = True              neg : pos    =      9.8 : 1.0
         contains(bears) = True              neg : pos    =      9.8 : 1.0
      contains(captures) = True              pos : neg    =      9.7 : 1.0
         contains(waste) = True              neg : pos    =      9.5 : 1.0
          contains(thin) = True              neg : pos    =      9.5 : 1.0
          contains(loud) = True              neg : pos    =      9.5 : 1.0
        contains(tender) = True              pos : neg    =      9.5 : 1.0
For pos
pos precision: 0.7346938775510204
pos recall: 0.7725321888412017
pos F-measure: 0.7531380753138076
For neg
neg precision: 0.792156862745098
neg recall: 0.7565543071161048
neg F-measure: 0.7739463601532567
['"Great people I adore them.', 'Had a girls with my sister and aunt and they di
d an excellent job on our nails.', 'Also had a wax and it felt so great no pain
very gentle.', 'I highly recommend to check them out."', '"Oh, my.', 'The bacon
egg and cheese biscuit was off the charts excellent.', "A perfectly fried egg, m
elty cheese and a bunch of bacon on top of a biscuit unlike any that I've enjoye
d before.", 'Different and a little grainy and very, very buttery.', 'Coffee was
', 'The bread was ridiculous, super soft with a slight crunch that and seemed to
 almost melt in your mouth.', "I did some research and found out that this bread
 comes straight from a New Orleans based bakery that's been in business for over
 a 100-years and I can see why.", 'Next time Dirty Cajun Fries are in my future.
']
```