In [1]:

```
import pandas as pd
import numpy as np
import collections
import pickle
import functools

# creating a global cache object for optimizing probability calculations
cache_dict = {}
```

Pre processing and tokenizing reviews (Slow Process: takes more than a minute)

In [2]:

```
%%time
import os
import string
import re

neg_path = './Data/neg/'
neg_files = os.listdir(neg_path)

pos_path = './Data/pos/'
pos_files = os.listdir(pos_path)

dataset = pd.DataFrame(columns=['review','polarity'])

for file_name in neg_files:
    try:
        fp = open(neg_path + file_name,'r')
        review_data = fp.read()
        review_class = 'neg'
        dataset = dataset.append(pd.Series([review_data,review_class], index=dataset.co
lumns),ignore_index=True)
        fp.close()
    except:
        # try except to deal with error in file reading due to codec issues
        pass

for file_name in pos_files:
    try:
        fp = open(pos_path + file_name,'r')
        review_data = fp.read()
        review_class = 'pos'
        dataset = dataset.append(pd.Series([review_data,review_class], index=dataset.co
lumns),ignore_index=True)
        fp.close()
    except:
        # try except to deal with error in file reading due to codec issues
        pass

# Basic cleaning
def cleanAndTokenize(review):
    # removing punctuations
    non_punc_words = "".join([character for character in review if character not in str
ing.punctuation])

    non_punc_words = non_punc_words.strip()

    # tokenizing reviews
    list_of_token = re.split('\W+',non_punc_words)

    list_of_token = set(list_of_token)

    list_of_token = list(list_of_token)

    return list_of_token

# converting words to lower case.
dataset['review'] = dataset['review'].apply(lambda review : cleanAndTokenize(str(review
).lower()))

vocabulary_list = dict()
```

```
for index,data_row in dataset.iterrows():
    for word in data_row['review']:
        if word in vocabulary_list:
            vocabulary_list[word].append(index)
        else:
            vocabulary_list[word] = [index]

# removing rare words
vocab_list = list(vocabulary_list.keys())
removed_words = list()
for word in vocab_list:
    if len(vocabulary_list[word]) < 5:
        removed_words.append(word)
        vocabulary_list.pop(word)

print(dataset.head())
```

```
                                              review polarity
0  [ashton, kid, with, why, very, rescue, most, s...      neg
1  [formal, violent, with, crazy, zsigmond, openi...      neg
2  [near, 108, sir, preferring, acting, versions,...      neg
3  [practically, with, countless, why, playing, g...      neg
4  [with, rappers, gangsters, startbr, five, acti...      neg
Wall time: 9min 12s
```

In [3]:

```
# all the functions probability functions
def get_probability_of_word(word,vocab_dict,dataset_with_label):
    word = word.lower()
    probability_word = 0.0

    if word in vocab_dict:
        if 'P["'+word+'"]' in cache_dict:
            probability_word = cache_dict['P["'+word+'"]']
            return probability_word

        word_count_doc = len(vocab_dict[word])
        probability_word = word_count_doc/dataset_with_label.shape[0]

        # caching results
        cache_dict['P["'+word+'"]'] = probability_word

    return probability_word

def get_probability_of_sentiment(word,vocab_dict,dataset_with_label,alpha=0):
    word = word.lower()

    if 'P["'+word+'|Positive"]' in cache_dict:
            probability_word = tuple([cache_dict['P["'+word+'|Positive"]'],cache_dict[
'P["'+word+'|Negative"]']])
            return probability_word

    probability_word = tuple([0,0])

    if word in vocab_dict:
        polarity_list = dataset_with_label.loc[vocab_dict[word],'polarity'].values
        class_count = dict(collections.Counter(polarity_list))

        if 'pos' not in class_count:
            class_count['pos'] = 0

        if 'neg' not in class_count:
            class_count['neg'] = 0

        # alpha is smoothing parameter
        probability_word = ((class_count['pos'] + alpha)/(len(dataset_with_label[datase
t_with_label['polarity'] == 'pos']) + 2 * alpha),
                            (class_count['neg'] + alpha)/(len(dataset_with_label[dataset_wi
th_label['polarity'] == 'neg']) + 2 * alpha))

        # caching results
        cache_dict['P["'+word+'|Positive"]'] = probability_word[0]
        cache_dict['P["'+word+'|Negative"]'] = probability_word[1]
    else:
        # alpha is smoothing parameter
        probability_word = ((alpha)/(len(dataset_with_label[dataset_with_label['polarit
y'] == 'pos']) + 2 * alpha),
                            (alpha)/(len(dataset_with_label[dataset_with_label['polarity']
== 'neg']) + 2 * alpha))

        # caching results
        cache_dict['P["'+word+'|Positive"]'] = probability_word[0]
        cache_dict['P["'+word+'|Negative"]'] = probability_word[1]

    return probability_word
```

Divide the dataset as train, development and test.

In [4]:

```
%%time
train, development, test = np.split(dataset.sample(frac=1, random_state=5), [int(.6*len
(dataset)), int(.8*len(dataset))])

print('size of train data set: ',train.shape[0])
print(train.head())
print('*'*100)
print('size of development data set: ',development.shape[0])
print(development.head())
print('*'*100)
print('size of test data set: ',test.shape[0])
print(test.head())
```

```
size of train data set:  30037
                                                  review polarity
991    [worst, everything, redeemable, entire, even, ...      neg
18653  [scary, believe, with, killing, will, at, even...      neg
6418   [worst, primitive, ends, mention, with, everyt...      neg
30937  [imdb, with, why, very, score, other, unusuall...      pos
11691  [scary, pg13, why, throughout, other, its, ent...      neg
**************************************************************************
**************************
size of development data set:  10012
                                                  review polarity
29979  [with, week, filler, remedied, everything, stu...      pos
35758  [sticks, showy, pace, allout, fruitful, with, ...      pos
12007  [with, scenes, acting, lesbian, entire, exactl...      neg
27532  [a, agenda, next, fix, some, one, own, more, g...      pos
47905  [employed, everything, gen, hhehebr, cinema, a...      pos
**************************************************************************
**************************
size of test data set:  10013
                                                  review polarity
49358  [a, not, with, birds, very, liked, shows, refr...      pos
42066  [crazy, shows, other, its, ensemble, loud, fis...      pos
45306  [perfect, with, perfectly, poor, very, playing...      pos
25279  [with, very, at, pros, leads, even, piling, po...      pos
7302   [worst, daytime, with, poor, rewarded, ship, w...      neg
Wall time: 180 ms
```

Build a vocabulary as list.

['the' 'I' 'happy' …] You may omit rare words for example if the occurrence is less than five times A reverse index as the key value might be handy {"the": 0, "I":1, "happy":2 , … }

In [5]:

```
print(list(vocabulary_list.keys())[0:5])
```

```
['ashton', 'kid', 'with', 'why', 'very']
```

Probability of the occurrence (P[“the”] = num of documents containing ‘the’ / num of all documents)

In [6]:

```
calc_prob_of = 'the'
print('P["'+calc_prob_of+'"] = %1.4f' % get_probability_of_word(calc_prob_of,vocabulary
_list,dataset))
cache_dict = {}
```

```
P["the"] = 0.9911
```

Conditional probability based on the sentiment P[“the” | Positive] = # of positive documents containing “the” / num of all positive review documents

In [7]:

```
calc_prob_of = 'the'
prob_values = get_probability_of_sentiment(calc_prob_of,vocabulary_list,dataset)
print('P["'+calc_prob_of+'|Positive"] = %1.4f' % prob_values[0])
print('P["'+calc_prob_of+'|Negative"] = %1.4f' % prob_values[1])
cache_dict = {}
```

```
P["the|Positive"] = 0.9899
P["the|Negative"] = 0.9922
```

Calculate accuracy using dev dataset (Very slow process: takes more than couple of minutes!)

In [8]:

```
%%time
vocabulary_list_train = dict()

cache_dict = {}

for index,data_row in train.iterrows():
    for word in data_row['review']:
        if word in vocabulary_list_train:
            vocabulary_list_train[word].append(index)
        else:
            vocabulary_list_train[word] = [index]

# removing rare words
vocab_list = list(vocabulary_list_train.keys())
removed_words_train = list()
for word in vocab_list:
    if len(vocabulary_list_train[word]) < 5:
        removed_words_train.append(word)
        vocabulary_list_train.pop(word)

accuracy_list = []

prob_class_pos = len(train[train['polarity'] == 'pos'])/train.shape[0]
prob_class_neg = len(train[train['polarity'] == 'neg'])/train.shape[0]

for index,rowdata in development.iterrows():
    positive_prob = []
    negative_prob = []
    for word in rowdata['review']:
        prob_values = get_probability_of_sentiment(word,vocabulary_list_train,train)
        positive_prob.append(prob_values[0])
        negative_prob.append(prob_values[1])

    pos_probability = functools.reduce(lambda x, y: x*y,positive_prob)
    neg_probability = functools.reduce(lambda x, y: x*y,negative_prob)

    pos_probability *= prob_class_pos
    neg_probability *= prob_class_neg

    if pos_probability > neg_probability:
        accuracy_list.append(rowdata['polarity'] == 'pos')
    else:
        accuracy_list.append(rowdata['polarity'] == 'neg')

print('Accuracy with respect to development dataset: %.2f' % ((sum(accuracy_list) / len
(accuracy_list)) * 100),'%')
```

```
Accuracy with respect to development dataset: 54.64 %
Wall time: 9min 26s
```

Conduct five fold cross validation (Very slow process: takes more than 15 minutes!)

In [9]:

```
%%time
fold1, fold2, fold3, fold4, fold5 = np.split(dataset.sample(frac=1, random_state=3), [i
nt(.2*len(dataset)), int(.4*len(dataset)), int(.6*len(dataset)), int(.8*len(dataset))])
cache_dict = {}
folds = [fold1, fold2, fold3, fold4, fold5]
for _ in range(5):
    test_dataset = folds.pop(0)
    train_dataset = pd.concat(folds)
    folds.append(test_dataset)

    vocabulary_list_train = dict()

    cache_dict = {}

    for index,data_row in train_dataset.iterrows():
        for word in data_row['review']:
            if word in vocabulary_list_train:
                vocabulary_list_train[word].append(index)
            else:
                vocabulary_list_train[word] = [index]

    # removing rare words
    vocab_list = list(vocabulary_list_train.keys())
    removed_words_train = list()
    for word in vocab_list:
        if len(vocabulary_list_train[word]) < 5:
            removed_words_train.append(word)
            vocabulary_list_train.pop(word)

    accuracy_list = []
    prob_class_pos = len(train_dataset[train_dataset['polarity'] == 'pos'])/train_datas
et.shape[0]
    prob_class_neg = len(train_dataset[train_dataset['polarity'] == 'neg'])/train_datas
et.shape[0]

    for index,rowdata in test_dataset.iterrows():
        positive_prob = []
        negative_prob = []
        for word in rowdata['review']:
            prob_values = get_probability_of_sentiment(word,vocabulary_list_train,train
_dataset)
            positive_prob.append(prob_values[0])
            negative_prob.append(prob_values[1])

        pos_probability = functools.reduce(lambda x, y: x*y,positive_prob)
        neg_probability = functools.reduce(lambda x, y: x*y,negative_prob)

        pos_probability *= prob_class_pos
        neg_probability *= prob_class_neg

        if pos_probability > neg_probability:
            accuracy_list.append(rowdata['polarity'] == 'pos')
        else:
            accuracy_list.append(rowdata['polarity'] == 'neg')

    print('Accuracy with respect to development dataset: %.2f' % ((sum(accuracy_list) /
len(accuracy_list)) * 100),'%')
```

```
    del train_dataset
    del test_dataset
```

```
Accuracy with respect to development dataset: 55.64 %
Accuracy with respect to development dataset: 55.34 %
Accuracy with respect to development dataset: 54.37 %
Accuracy with respect to development dataset: 55.06 %
Accuracy with respect to development dataset: 54.67 %
Wall time: 1h 29s
```

Compare the effect of Smoothing (Very slow process: takes more than an hour!)

In [10]:

```
%%time
for aplha_num in range(1,20):
    vocabulary_list_train = dict()

    cache_dict = {}

    for index,data_row in train.iterrows():
        for word in data_row['review']:
            if word in vocabulary_list_train:
                vocabulary_list_train[word].append(index)
            else:
                vocabulary_list_train[word] = [index]

    # removing rare words
    vocab_list = list(vocabulary_list_train.keys())
    removed_words_train = list()
    for word in vocab_list:
        if len(vocabulary_list_train[word]) < 5:
            removed_words_train.append(word)
            vocabulary_list_train.pop(word)

    accuracy_list = []

    prob_class_pos = len(train[train['polarity'] == 'pos'])/train.shape[0]
    prob_class_neg = len(train[train['polarity'] == 'neg'])/train.shape[0]

    for index,rowdata in development.iterrows():
        positive_prob = []
        negative_prob = []
        for word in rowdata['review']:
            prob_values = get_probability_of_sentiment(word,vocabulary_list_train,train
,aplha_num)
            positive_prob.append(prob_values[0])
            negative_prob.append(prob_values[1])

        pos_probability = functools.reduce(lambda x, y: x*y,positive_prob)
        neg_probability = functools.reduce(lambda x, y: x*y,negative_prob)

        pos_probability *= prob_class_pos
        neg_probability *= prob_class_neg

        if pos_probability > neg_probability:
            accuracy_list.append(rowdata['polarity'] == 'pos')
        else:
            accuracy_list.append(rowdata['polarity'] == 'neg')

    print('Accuracy with respect to development dataset: %.2f' % ((sum(accuracy_list) /
len(accuracy_list)) * 100),'% alpha = ',aplha_num)
```

```
Accuracy with respect to development dataset: 79.96 % alpha =  1
Accuracy with respect to development dataset: 80.12 % alpha =  2
Accuracy with respect to development dataset: 80.19 % alpha =  3
Accuracy with respect to development dataset: 80.26 % alpha =  4
Accuracy with respect to development dataset: 80.34 % alpha =  5
Accuracy with respect to development dataset: 80.32 % alpha =  6
Accuracy with respect to development dataset: 80.34 % alpha =  7
Accuracy with respect to development dataset: 80.39 % alpha =  8
Accuracy with respect to development dataset: 80.45 % alpha =  9
Accuracy with respect to development dataset: 80.37 % alpha =  10
Accuracy with respect to development dataset: 80.41 % alpha =  11
Accuracy with respect to development dataset: 80.44 % alpha =  12
Accuracy with respect to development dataset: 80.44 % alpha =  13
Accuracy with respect to development dataset: 80.41 % alpha =  14
Accuracy with respect to development dataset: 80.40 % alpha =  15
Accuracy with respect to development dataset: 80.44 % alpha =  16
Accuracy with respect to development dataset: 80.41 % alpha =  17
Accuracy with respect to development dataset: 80.44 % alpha =  18
Accuracy with respect to development dataset: 80.50 % alpha =  19
Wall time: 2h 50min 31s
```

Derive Top 10 words that predicts positive and negative class P[Positive| word] Using the test dataset (Slow Process: takes about 13 mins)

In [11]:

```
%%time
cache_dict = {}
positive_prob = []
negative_prob = []

for word in vocabulary_list:
    prob_values = get_probability_of_sentiment(word,vocabulary_list_train,dataset)
    positive_prob.append(prob_values[0])
    negative_prob.append(prob_values[1])

data_dict = {'word':pd.Series(list(vocabulary_list.keys())), 'pos_prob':pd.Series(posit
ive_prob), 'neg_prob':pd.Series(negative_prob)}
probability_dataframe = pd.DataFrame(data_dict)

# top 10 words
probability_dataframe.sort_values(by='pos_prob',ascending=False,inplace=True)
print('Top 10 positive words: \n',probability_dataframe['word'].iloc[0:10].values)
probability_dataframe.sort_values(by='neg_prob',ascending=False,inplace=True)
print('Top 10 negative words:  \n',probability_dataframe['word'].iloc[0:10].values)
```

```
Top 10 positive words: 
 ['the' 'and' 'a' 'of' 'to' 'is' 'this' 'in' 'it' 'that']
Top 10 negative words: 
 ['the' 'a' 'and' 'to' 'of' 'this' 'is' 'in' 'it' 'that']
Wall time: 7min 13s
```

Use the optimal hyperparameters you found in the step e, and use it to calculate the final accuracy. Use five fold cross validation for final accuracy

In [12]:

```
%%time
vocabulary_list_train = dict()

alpha_param = 1

cache_dict = {}

for index,data_row in train.iterrows():
    for word in data_row['review']:
        if word in vocabulary_list_train:
            vocabulary_list_train[word].append(index)
        else:
            vocabulary_list_train[word] = [index]

# removing rare words
vocab_list = list(vocabulary_list_train.keys())
removed_words_train = list()
for word in vocab_list:
    if len(vocabulary_list_train[word]) < 5:
        removed_words_train.append(word)
        vocabulary_list_train.pop(word)

accuracy_list = []

prob_class_pos = len(train[train['polarity'] == 'pos'])/train.shape[0]
prob_class_neg = len(train[train['polarity'] == 'neg'])/train.shape[0]

for index,rowdata in test.iterrows():
    positive_prob = []
    negative_prob = []
    for word in rowdata['review']:
        prob_values = get_probability_of_sentiment(word,vocabulary_list_train,train,alp
ha_param)
        positive_prob.append(prob_values[0])
        negative_prob.append(prob_values[1])

    pos_probability = functools.reduce(lambda x, y: x*y,positive_prob)
    neg_probability = functools.reduce(lambda x, y: x*y,negative_prob)

    pos_probability *= prob_class_pos
    neg_probability *= prob_class_neg

    if pos_probability > neg_probability:
        accuracy_list.append(rowdata['polarity'] == 'pos')
    else:
        accuracy_list.append(rowdata['polarity'] == 'neg')

print('Final accuracy with respect to test dataset: %.2f' % ((sum(accuracy_list) / len(
accuracy_list)) * 100),'%')
```

```
Final accuracy with respect to test dataset: 80.05 %
Wall time: 8min 52s
```