In [1]:

```
"""Question 1"""
```

Out[1]:

```
'Question 1'
```

In [2]:

```
import pandas as pd
import numpy as np
import operator
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import normalize
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
import string
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
#a:  Parse the following data
amazon_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/ser
                          sep = ".   ", header = None, names = ["review", "value"])
amazon_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

Out[3]:

	review	value
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value	1
2	Great for the jawbone	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great	1

In [4]:

```
yelp_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/senti
                          sep = ".   ", header = None, names = ["review", "value"])
yelp_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
```

Out[4]:

	review	value
0	Wow... Loved this place	1
1	Crust is not good	0
2	Not tasty and the texture was just nasty	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

In [5]:

```
imdb_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/senti
                          sep = ".  ", header = None, names = ["review", "value"])
imdb_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
```

Out[5]:

	review	value
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1

In [6]:

```
print ("Amazon     |    Yelp     |    IMDB")
print (amazon_data.shape, "|", yelp_data.shape, "|", imdb_data.shape)
```

```
Amazon     |    Yelp     |    IMDB
(1000, 2) | (1000, 2) | (1000, 2)
```

In [7]:

```
# Label Balance
"""Given that the shape of all three text files includes 100 reviews with either a 0 or 1 v
```

Out[7]:

```
'Given that the shape of all three text files includes 100 reviews with eith
er a 0 or 1 value and we know that sum of the values are 500, we know that t
he labels must be balanced.'
```

In [8]:

```
#b: Preprocessing of the data

## Remove stopwords
def strip_punctuation(line):
    result = ""
    for c in line:
        if c not in string.punctuation:
            result += c
    return result

## Lemmatization of all the words
def lemmatize_line(line):
    lemma = WordNetLemmatizer()
    word_list = nltk.word_tokenize(line)
    lemmatized_output = ' '.join([lemma.lemmatize(w, pos='a') for w in word_list])
    word_list = nltk.word_tokenize(lemmatized_output)
    lemmatized_output = ' '.join([lemma.lemmatize(w, pos='v') for w in word_list])
    return lemmatized_output

## Strip stop words
def strip_stop_words(line):
    word_list = nltk.word_tokenize(line)
    filtered_words = ' '.join([word for word in word_list if word not in stopwords.words('e
    return filtered_words
```

In [9]:

```
def preprocess_data(raw_data):
    data = raw_data.apply(lambda x: x.astype(str).str.lower())
    pd.to_numeric(data['value'])
    data['review'].astype('str')
    data['review'] = data['review'].apply(lambda x: lemmatize_line(x))
    data['review'] = data['review'].apply(lambda x: strip_stop_words(x))
    data['review'] = data['review'].apply(lambda x: strip_punctuation(x))
    return data
```

In [10]:

```
amazon_data_preprocess = preprocess_data(amazon_data)
amazon_data_preprocess.head()
```

Out[10]:

	review	value
0	way plug us unless go converter	0
1	good case excellent value	1
2	great jawbone	1
3	tie charger conversations last 45 minutesmajor...	0
4	mic great	1

In [11]:

```
yelp_data_preprocess = preprocess_data(yelp_data)
yelp_data_preprocess.head()
```

Out[11]:

	review	value
0	wow love place	1
1	crust good	0
2	tasty texture nasty	0
3	stop late may bank holiday rick steve recommen...	1
4	selection menu great price	1

In [12]:

```
imdb_data_preprocess = preprocess_data(imdb_data)
imdb_data_preprocess.head()
```

Out[12]:

	review	value
0	slowmoving aimless movie distress drift yo...	0
1	sure lose flat character audience nearly hal...	0
2	attempt artiness black white clever camera an...	0
3	little music anything speak	0
4	best scene movie gerardo try find song keep ru...	1

In [13]:

```
#c: Split train-test as 80:20
def split_data(data):
    tr, te = train_test_split(data, test_size=0.2)
    return tr, te
```

In [14]:

```
amazon_train, amazon_test = split_data(amazon_data_preprocess)
yelp_train, yelp_test = split_data(yelp_data_preprocess)
imdb_train, imdb_test = split_data(imdb_data_preprocess)
```

In [15]:

```
train_data = pd.DataFrame()
train_data = pd.concat([amazon_train, imdb_train], ignore_index = True)
train_data = pd.concat([train_data, yelp_train], ignore_index = True)
train_data.shape
```

Out[15]:

```
(2400, 2)
```

In [16]:

```
test_data = pd.DataFrame()
test_data = pd.concat([amazon_test, imdb_test], ignore_index = True)
test_data = pd.concat([test_data, yelp_test], ignore_index = True)
test_data.shape
```

Out[16]:

```
(600, 2)
```

In [17]:

```
x_train, y_train = train_data['review'], train_data['value']
x_test, y_test = test_data['review'], test_data['value']
print (x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(2400,) (2400,) (600,) (600,)
```

In [18]:

```
#d: build a dictionary of unique words for training set
train_vectorizer = CountVectorizer()
train_x_bow = train_vectorizer.fit_transform(x_train).todense()
test_vectorizer = CountVectorizer(vocabulary=train_vectorizer.get_feature_names())
test_x_bow = test_vectorizer.fit_transform(x_test).todense()
```

In [19]:

```
"""Above we have vectorized training set first, followed by limiting the vocubulary for tes
```

Out[19]:

```
'Above we have vectorized training set first, followed by limiting the vocub
ulary for test set as otherwise the test-set might contains words which will
be absent in training set, and we will get incorrect feature vectors'
```

In [20]:

```
# Report feature vectors of any two reviews in the training set
print(x_train[4])
print(train_x_bow[4])
print(x_train[21])
print(train_x_bow[21])
```

```
echo problem  very unsatisfactor
[[0 0 0 ... 0 0 0]]
 warranty  problems reoccurebottom line  put money somewhere else  cingular
support
[[0 0 0 ... 0 0 0]]
```

In [21]:

```
#e: Post-processing
train_x_bow_normal = normalize(train_x_bow, norm='l1')
test_x_bow_normal = normalize(test_x_bow, norm='l1')
```

In [22]:

```
"""Here, I have used L1 normalization because the reviews are qualitative so the rendundanc
```

Out[22]:

```
"Here, I have used L1 normalization because the reviews are qualitative so t
he rendundancy is unavoidable and there's no absolute measurement."
```

In [23]:

```
#f: Sentiment Analysis using Logistic regression and Naive-based (Gaussian, Bernoulli and M
def sentiment_prediction(x_tr, y_tr, x_te, y_te):

    lr_clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').
    lr_score = lr_clf.score(x_te, y_te)
    print("Logistic regression accuracy: {}".format(lr_score))
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, lr_clf.predict(x_te)))

    b_nb = BernoulliNB()
    b_fit = b_nb.fit(x_tr, y_tr)
    b_nb_score = b_fit.score(x_te, y_te)
    print("Accuracy of Naive Bayes Classifier with Bernoulli prior: {}".format(b_nb_score))
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, b_fit.predict(x_te)))

    gaussian_nb = GaussianNB()
    gaussian_nb.fit(x_tr, y_tr)
    gaussian_nb_score = gaussian_nb.score(x_te, y_te)
    print("Accuracy of Naive Bayes Classifier with Gaussian prior: {}".format(gaussian_nb_s
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, gaussian_nb.predict(x_te)))

    return lr_score, gaussian_nb_score, b_nb_score, lr_clf.coef_
```

In [24]:

```
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_bow_normal
                                                                                      test_x
```

```
Logistic regression accuracy: 0.7883333333333333
Confussion Matrix:
 [[255  66]
 [ 61 218]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.8133333333333334
Confussion Matrix:
 [[262  59]
 [ 53 226]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6883333333333334
Confussion Matrix:
 [[260  61]
 [126 153]]
```

In [25]:

```
vocabulary_bow = train_vectorizer.get_feature_names()
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
weight_vector
```

Out[25]:

```
[('great', 3.342498970911084),
 ('love', 2.297285972463214),
 ('nice', 1.902211561012528),
 ('excellent', 1.7382161891503407),
 ('amaze', 1.5418266737078257),
 ('good', 1.5301828050738429),
 ('best', 1.484047481851821),
 ('delicious', 1.3159122566579229),
 ('fantastic', 1.230057723356934),
 ('well', 1.219354117628204),
 ('awesome', 1.1642286385405223),
 ('wonderful', 1.087780198567724),
 ('perfect', 1.0828387595913571),
 ('beautiful', 1.04841777935078),
 ('friendly', 1.0207149155839712),
 ('price', 0.9733547293305825),
 ('enjoy', 0.9668645452318055),
 ('comfortable', 0.9379343418781981),
```

In [26]:

```
"""I see Burnoulli for NBC gives me the best result, followed by Logistic Regression and th
```

Out[26]:

```
'I see Burnoulli for NBC gives me the best result, followed by Logistic Regr
ession and the worst is Gaussian NBC'
```

In [27]:

```
#g: N-gram model
train_vectorizer_2gram = CountVectorizer(ngram_range=(2, 2))
train_x_2gram = train_vectorizer_2gram.fit_transform(x_train).todense()
test_vectorizer_2gram = CountVectorizer(ngram_range=(2, 2), vocabulary=train_vectorizer_2gr
test_x_2gram = test_vectorizer_2gram.fit_transform(x_test).todense()
print(x_train[4])
print(train_x_2gram[4])
print(x_train[21])
print(train_x_2gram[21])
```

```
echo problem  very unsatisfactor
[[0 0 0 ... 0 0 0]]
 warranty  problems reoccurebottom line  put money somewhere else  cingular
support
[[0 0 0 ... 0 0 0]]
```

In [28]:

```
train_x_2gram_normal = normalize(train_x_2gram, norm='l1')
test_x_2gram_normal = normalize(test_x_2gram, norm='l1')
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_2gram_norm
                                                                                          y_trai
```

```
Logistic regression accuracy: 0.5966666666666667
Confussion Matrix:
 [[114 207]
 [ 35 244]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5966666666666667
Confussion Matrix:
 [[109 212]
 [ 30 249]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6683333333333333
Confussion Matrix:
 [[282  39]
 [160 119]]
```

In [29]:

```
vocabulary_ng = train_vectorizer_2gram.get_feature_names()
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
weight_vector
```

Out[29]:

```
[('work great', 1.2007221477664503),
 ('highly recommend', 1.061456145145739),
 ('great phone', 0.836510713908666),
 ('food good', 0.7456916059201351),
 ('one best', 0.7403189151851477),
 ('love place', 0.7399525083544034),
 ('good price', 0.6681850208031861),
 ('great product', 0.6499576092806894),
 ('excellent product', 0.6330740446141688),
 ('easy use', 0.6326868650730371),
 ('really good', 0.5986319213213821),
 ('good product', 0.5864949812650904),
 ('love phone', 0.5723222224750062),
 ('great price', 0.5681441125462072),
 ('pretty good', 0.5567358109681317),
 ('food delicious', 0.540999000359394),
 ('great deal', 0.533278800774241),
 ('nt disappoint', 0.5203255508206126),
```

In [30]:

```
"""For this case, I see the output of all the process is similar, but wierdly Gaussian perf
```

Out[30]:

```
'For this case, I see the output of all the process is similar, but wierdly
Gaussian performs slightly better'
```

In [31]:

```
#h : PCA Analysis using SVD for Bag of Word Model
p,n = np.shape(train_x_bow_normal)
cov_matrix = np.dot(train_x_bow_normal.T, train_x_bow_normal)/(p-1)
u, s, vh = np.linalg.svd(cov_matrix, full_matrices=True)
```

In [32]:

```
# PCA Analysis using SVD for N-gram model
p2,n2 = np.shape(train_x_2gram_normal)
cov_matrix2 = np.dot(train_x_2gram_normal.T, train_x_2gram_normal)/(p-1)
u2, s2, vh2 = np.linalg.svd(cov_matrix2, full_matrices=True)
```

In [33]:

```
# Dimension=10
train_x_10 = np.dot(train_x_bow_normal, u[:,:10])
test_x_10 = np.dot(test_x_bow_normal, u[:,:10])
print ("PCA dim 10 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_10, y_train

train_x_10_ng = np.dot(train_x_2gram_normal, u2[:,:10])
test_x_10_ng = np.dot(test_x_2gram_normal, u2[:,:10])
print ("PCA dim 10 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_x
```

```
PCA dim 10 for Bag of Word Model
Logistic regression accuracy: 0.635
Confussion Matrix:
 [[234  87]
 [132 147]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6066666666666667
Confussion Matrix:
 [[219 102]
 [134 145]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6233333333333333
Confussion Matrix:
 [[268  53]
 [173 106]]
PCA dim 10 for Ngram Model
Logistic regression accuracy: 0.48333333333333334
Confussion Matrix:
 [[ 12 309]
 [  1 278]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5216666666666666
Confussion Matrix:
 [[198 123]
 [164 115]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.55
Confussion Matrix:
 [[315   6]
 [264  15]]
```

In [34]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 2.8577505755728434), ('1010', 1.7759771532555801), ('110', 1.423135
02364502), ('15', -0.6693773261571053), ('13', -0.9502073859538739), ('12',
-1.2898382254225256), ('11', -1.6560349037860573), ('10', -1.672427221606093
6), ('1199', -2.353145926039954), ('100', -3.1641965961671294)]
N-gram model
[('010 grade', 1.301424693125726), ('10 movie', 1.1854864740915152), ('10 1
0', 0.7551715058608989), ('10 feet', 0.750863293325864), ('10 oyvey', 0.3577
0008633371153), ('10 minutes', 0.32908254476652155), ('10 plus', 0.262841373
44038633), ('10 simply', -0.37613923757685475), ('10 save', -1.0087971191978
81), ('10 110', -1.6805937767361494)]
```

In [37]:

```
# Dimension=50
train_x_50 = np.dot(train_x_bow_normal, u[:,:50])
test_x_50 = np.dot(test_x_bow_normal, u[:,:50])
print ("PCA dim 50 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_50, y_train

train_x_50_ng = np.dot(train_x_2gram_normal, u2[:,:50])
test_x_50_ng = np.dot(test_x_2gram_normal, u2[:,:50])
print ("PCA dim 50 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_x
```

```
PCA dim 50 for Bag of Word Model
Logistic regression accuracy: 0.6883333333333334
Confussion Matrix:
 [[231  90]
 [ 97 182]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6366666666666667
Confussion Matrix:
 [[242  79]
 [139 140]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.605
Confussion Matrix:
 [[153 168]
 [ 69 210]]
PCA dim 50 for Ngram Model
Logistic regression accuracy: 0.5016666666666667
Confussion Matrix:
 [[ 26 295]
 [  4 275]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5466666666666666
Confussion Matrix:
 [[212 109]
 [163 116]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.5566666666666666
Confussion Matrix:
 [[309  12]
 [254  25]]
```

In [38]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 3.0175075444588986), ('2007', 2.1616212233553), ('20th', 1.78239967
6560226), ('1010', 1.7084880952585775), ('15lb', 1.5640357645893908), ('11
0', 1.5339136787718948), ('40min', 1.3775368316401804), ('2000', 1.175990972
34076), ('34ths', 0.9874919338561612), ('350', 0.9604229230422221), ('45',
0.9537271698196184), ('5320', 0.9509793983742273), ('42', 0.907159607272713
8), ('70s', 0.8205158174925946), ('1979', 0.7618382864541814), ('5020', 0.74
94372122304447), ('2mp', 0.6052035005921853), ('24', 0.5303940609376162),
('2006', 0.4958316457191242), ('3o', 0.49216090494015213), ('70000', 0.38949
97871463807), ('325', 0.33827299203740013), ('1995', 0.28781694644633266),
('2005', 0.22129709031873795), ('1947', 0.1822031267465752), ('20the', 0.16
901724279099792), ('70', 0.11887847749019537), ('25', 0.05392323288165849),
('1948', -0.0710876072905581), ('1998', -0.1098688603137161), ('23', -0.1105
2739653559647), ('30', -0.1359252357881056), ('1980', -0.37910378530586036),
('2160', -0.4467284834279109), ('17', -0.6590020200483581), ('18th', -0.7318
015194425378), ('5year', -0.7518320114474671), ('18', -0.766736016361032),
('15', -0.7750780479749669), ('13', -0.8796997258714572), ('35', -0.88160591
07661472), ('40', -0.8896548360329873), ('12', -1.3426266178223056), ('192
8', -1.3870624619430134), ('10', -1.6238857678880338), ('11', -1.64565506189
41097), ('20', -1.7275498339443063), ('510', -1.895987251183887), ('1199', -
2.5014903483341366), ('100', -3.0434084506771555)]
N-gram model
[('010 grade', 1.3013849907268018), ('10 movie', 1.1794210903601388), ('13 b
uck', 0.77690648844648), ('13 megapixels', 0.7675158561189311), ('10 10', 0.
7572655495461976), ('10 feet', 0.7503144716139528), ('18 months', 0.66143187
8127891), ('10 star', 0.6505509661249359), ('15lb piece', 0.614520343653597
3), ('10 years', 0.582825359744074), ('2005 buy', 0.5660864676106206), ('194
7 masterpiece', 0.559542703734381), ('100 time', 0.5579954205973328), ('100
recommend', 0.5491969764510868), ('20 leave', 0.5290284385235178), ('12 meg
a', 0.5268455578653997), ('110 set', 0.5268455578653983), ('1199 sandwich',
0.5160537264320827), ('2007 every', 0.5071702287164133), ('1998 deep', 0.492
3223429890355), ('1995 monster', 0.4816989756845576), ('17 burger', 0.414910
452871482), ('2005 begin', 0.3923112604773328), ('10 oyvey', 0.3589625186873
818), ('12 minutes', 0.3554630701590764), ('10 minutes', 0.333238969443557
1), ('12 mile', 0.3298700880487148), ('10 plus', 0.26129384920178117), ('20
feet', 0.1496809144052962), ('20 minutes', 0.10578232783119595), ('1979 firs
t', 0.0881900847306004), ('15 minutes', 0.07178901948497245), ('12 ridiculou
s', 0.01947313249602486), ('10 simply', -0.3763591019550324), ('20th centur
y', -0.4475803549167901), ('20th 2005', -0.4475803549168037), ('20the cove
r', -0.4576153367190696), ('20 30', -0.4791193812310562), ('20 ca', -0.48169
897568455783), ('1980 experience', -0.49483311440377414), ('20 years', -0.51
42150622026452), ('12 hours', -0.5160537264320831), ('11 months', -0.5268455
578653983), ('110 scale', -0.526845557865399), ('18th century', -0.540764780
8736893), ('15 second', -0.6152671778127168), ('1948 quite', -0.892006270050
6257), ('10 save', -1.0109232834180493), ('10 time', -1.0195726668350913),
('10 110', -1.6763894351624153)]
```

In [39]:

```
# Dimension=100
train_x_100 = np.dot(train_x_bow_normal, u[:,:100])
test_x_100 = np.dot(test_x_bow_normal, u[:,:100])
print ("PCA dim 100 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_100, y_tra

train_x_100_ng = np.dot(train_x_2gram_normal, u2[:,:100])
test_x_100_ng = np.dot(test_x_2gram_normal, u2[:,:100])
print ("PCA dim 100 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_
```

```
PCA dim 100 for Bag of Word Model
Logistic regression accuracy: 0.7283333333333334
Confussion Matrix:
 [[243  78]
 [ 85 194]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6316666666666667
Confussion Matrix:
 [[239  82]
 [139 140]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6266666666666667
Confussion Matrix:
 [[152 169]
 [ 55 224]]
PCA dim 100 for Ngram Model
Logistic regression accuracy: 0.515
Confussion Matrix:
 [[ 35 286]
 [  5 274]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.55
Confussion Matrix:
 [[215 106]
 [164 115]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.5733333333333334
Confussion Matrix:
 [[307  14]
 [242  37]]
```

In [40]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 2.9834418998057752), ('2007', 2.167599341128274), ('absolutely', 1.
873229156828621 7), ('20th', 1.7393013703611717), ('1010', 1.717023232621198
3), ('15lb', 1.5479557435823645), ('110', 1.518133239512729), ('40min', 1.39
58184690452062), ('accessoryone', 1.2059869310839324), ('accident', 1.154519
796613082), ('2000', 1.1145959270282133), ('5320', 0.973337337236965), ('34t
hs', 0.9682212674411931), ('45', 0.9598660273249721), ('350', 0.950987404851
4296), ('42', 0.8718156847168762), ('70s', 0.8440325971574535), ('5020', 0.7
878417263875597), ('910', 0.7749260965204591), ('1979', 0.7439842558805747),
('accuse', 0.7074871823765152), ('accurate', 0.6380581179634662), ('2mp', 0.
5744384637057823), ('24', 0.519366714077008), ('3o', 0.4990231724183387),
('accolades', 0.49696301364022627), ('2006', 0.4856098465279725), ('accessib
le', 0.426715095157759), ('absolute', 0.4187971577467303), ('70000', 0.40546
563909125805), ('ache', 0.3940249310070008), ('accessable', 0.36584498715124
7), ('325', 0.3538826301230212), ('accent', 0.3404664863588703), ('80', 0.33
01218012394722), ('achievement', 0.3111013745755125), ('accessory', 0.306686
2012135552), ('1995', 0.2948191899869568), ('accomodate', 0.278264312576600
7), ('accountant', 0.25995555710947144), ('2005', 0.23346842352134967), ('19
47', 0.20170302984551328), ('20the', 0.17865901123008066), ('abroad', 0.0895
7494959354406), ('70', 0.07247535794685991), ('25', 0.05865042082135328),
('accordingly', 0.03872375531613293), ('accept', 0.028423210878726794), ('ab
ysmal', 0.010857051387616976), ('785', -0.052647237185990306), ('1948', -0.0
6293852692257758), ('23', -0.10115573003758613), ('absolutel', -0.1019929503
077234), ('1998', -0.10674042439710427), ('ability', -0.12458132911227182),
('8pm', -0.13272346298730736), ('30', -0.13821373748533808), ('accidentall
y', -0.152183526039955), ('abstruse', -0.19576147504367075), ('810', -0.2059
736112394467), ('abhor', -0.21497755516137548), ('absolutley', -0.2183582634
113116), ('acclaim', -0.227573664919918), ('ackerman', -0.229894683989415 6
8), ('access', -0.2729484985376612), ('academy', -0.2784279675264577), ('198
0', -0.3711787878545898), ('accommodations', -0.39988892163910034), ('2160',
-0.4075436307782216), ('8530', -0.5027651291972507), ('815pm', -0.5997631306
850566), ('acknowledge', -0.6301803310291424), ('17', -0.6490294654717448),
('ac', -0.6626448532191959), ('acceptable', -0.6675920052525571), ('18th', -
0.710867249644412), ('8125', -0.7444811165702646), ('5year', -0.745646011206
6449), ('15', -0.7462470994891878), ('18', -0.768565034177833), ('act', -0.7
909565784925332), ('abandon', -0.7976889606773196), ('90', -0.81370219872854
89), ('35', -0.8675332453307769), ('accord', -0.880269347565283), ('13', -0.
8994893794620034), ('40', -0.9053392169134851), ('750', -1.017177392371124
7), ('aailiyah', -1.1623520635328606), ('744', -1.3084119713255833), ('12',
-1.318211496006284), ('1928', -1.3704361062738248), ('10', -1.623572945244
9), ('11', -1.6479620383505782), ('20', -1.7168706234820474), ('95', -1.7921
449723196794), ('510', -1.8471745105937192), ('able', -1.9606004979449747),
('1199', -2.4797341911931294), ('100', -3.0651981132352843)]
N-gram model
[('010 grade', 1.304923397790976), ('10 movie', 1.1810600765349242), ('13 bu
ck', 0.7750674182541492), ('13 megapixels', 0.7659648554782152), ('10 10',
0.756031603169735), ('10 feet', 0.7522102359727986), ('18 months', 0.6605010
```

```
950238082), ('10 star', 0.6513963396767805), ('785 hot', 0.630981385385321
2), ('15lb piece', 0.6134691278575708), ('10 years', 0.5835707467865591),
('2005 buy', 0.5681846042516381), ('1947 masterpiece', 0.55864747910991),
('100 time', 0.556784146756771), ('100 recommend', 0.5500111410333852), ('20
leave', 0.5305742250289868), ('40min pass', 0.5300334502474656), ('12 mega',
0.5257310942520028), ('110 set', 0.5257310942520018), ('70 claim', 0.5185444
282460603), ('1199 sandwich', 0.5167673095347537), ('2007 every', 0.50887779
11080635), ('1998 deep', 0.4914969267478018), ('1995 monster', 0.48271150104
471383), ('24 hours', 0.46122345894767336), ('2160 tracfone', 0.449001783717
4799), ('30 40', 0.43217829024610677), ('ability pull', 0.420323667737197),
('17 burger', 0.41806552315392126), ('35 big', 0.4120664950007633), ('40 yea
rs', 0.39515153103769673), ('2005 begin', 0.39362587727639303), ('45 minute
s', 0.3927843302360884), ('95 garbage', 0.37416785906103317), ('510 maintai
n', 0.3705206690437831), ('10 oyvey', 0.36098642609070597), ('12 minutes',
0.3571136381441496), ('10 minutes', 0.3321442823056211), ('12 mile', 0.32872
32734824405), ('5020 pretty', 0.2736509425309856), ('34ths gristle', 0.26893
46886302015), ('10 plus', 0.2575512705786513), ('80 wonderful', 0.2175717355
009501), ('23 decent', 0.15404135611751438), ('20 feet', 0.1499833000610818
5), ('42 optimal', 0.13806460102768317), ('8pm start', 0.12452868965390543),
('20 minutes', 0.10319597866512915), ('1979 first', 0.08918116038106631),
('aailiyah pretty', 0.08872164014963929), ('15 minutes', 0.0697035690393703
5), ('ability phone', 0.0549134746956784), ('8530 blackberry', 0.01962649706
731371), ('12 ridiculous', 0.019459376044383733), ('80 flick', -0.0264516430
85599344), ('90 minutes', -0.10523431916619405), ('5year work', -0.115190728
19568894), ('70s grainy', -0.12017680355345685), ('ability dwight', -0.16601
250558286548), ('810 score', -0.20688021510086174), ('25 years', -0.21804223
257655578), ('abandon factory', -0.23070394431372948), ('30 min', -0.2350551
6506822274), ('5year old', -0.2470357162867931), ('90 food', -0.263234410024
7313), ('ability meld', -0.30619159803811885), ('40 handle', -0.315548525631
3202), ('45 minutesmajor', -0.32935718450913853), ('90 child', -0.3348375250
5852965), ('510 right', -0.3652732725003369), ('325 cellphone', -0.374558263
00582157), ('10 simply', -0.376382503011259), ('510 complaints', -0.38148731
59467417), ('350 headset', -0.4072081553838106), ('2mp pics', -0.42429974594
93747), ('40 minutes', -0.4259350665092096), ('20th century', -0.44900178371
74656), ('20th 2005', -0.44900178371747873), ('90 mainly', -0.45091217822719
76), ('20the cover', -0.4572941441226327), ('70000 time', -0.457785392149405
45), ('3o minutes', -0.45785427577875154), ('20 30', -0.47826689250644705),
('20 ca', -0.48271150104471405), ('30 minutes', -0.4863926632997367), ('1980
experience', -0.4958229540628439), ('20 years', -0.5158034144657175), ('12 h
ours', -0.5167673095347538), ('11 months', -0.5257310942520017), ('110 scal
e', -0.5257310942520024), ('18th century', -0.5418896032011234), ('15 secon
d', -0.6163893659656858), ('815pm fast', -0.6202990146821818), ('750 see', -
0.6369543208134447), ('744 plug', -0.8251923564996154), ('1948 quite', -0.88
89339310507927), ('10 save', -1.0076313975904052), ('10 time', -1.0209959187
181086), ('23 bar', -1.115423865765869), ('10 110', -1.6726809518236487)]
```

In [41]:

```
#i : Algorithm review
"""1) Bag of words using NBC with Burnoulli performs best. It might be because bag of words
```

Out[41]:

```
'1) Bag of words using NBC with Burnoulli performs best. It might be because
bag of words reserved all features of words and single word could represent
features better.'
```

In [42]:

```
"""2) The reason that PCA did not work well for this dataset might be because:
The word features are relatively evenly distributed in all features, which means the direct
Originally the dataset has around 2000 features. Reducing them to ~100 features reduced toc
```

Out[42]:

```
'2) The reason that PCA did not work well for this dataset might be becaus
e:\nThe word features are relatively evenly distributed in all features, whi
ch means the directions with highest variance cannot represent most informat
ion of the original dataset. So reducing dimensions will lose considerable p
art of original information.\nOriginally the dataset has around 2000 feature
s. Reducing them to ~100 features reduced too much information.'
```

In [43]:

```
"""3) We see that Bag of Words always perform better than N-gram model, for all cases, both
```

Out[43]:

```
'3) We see that Bag of Words always perform better than N-gram model, for al
l cases, both with and without PCA.'
```

In [ ]:

```

```