KUMAR GAURAV 20122065

Research part

Part 3

In [ ]:

```
import pandas as pd
import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import mlxtend
import sklearn.cluster as cluster
import sklearn.neighbors
import sklearn.metrics as metrics
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import string
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,confusion_matrix, roc_curve, auc, precisi
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
train_data = pd.read_csv('../input/emotions-dataset-for-nlp/train.txt',names=['sentence','e
test_data = pd.read_csv('../input/emotions-dataset-for-nlp/test.txt',names=['sentence','emo
val_data= pd.read_csv('../input/emotions-dataset-for-nlp/val.txt',names=['sentence','emotio
df = pd.concat([train_data,test_data, val_data])
print('Total data:',df.shape)
```

```
Total data: (20000, 2)
```

In [3]:

```
# Null Check
train_data.isnull().sum()
test_data.isnull().sum()
val_data.isnull().sum()
```

Out[3]:

```
sentence    0
emotion     0
dtype: int64
```

In [4]:

```
df = df.drop_duplicates(keep="first") # Drop duplicated data and reindex the data
df_reidx = df.reset_index(drop=True)
df_reidx.shape
```

Out[4]:

```
(19999, 2)
```

In [5]:

```
# convert the emotions to binary labels. love and joy emotions are "not-stressed ==1", and
df_reidx['label']=df_reidx['emotion'].replace({'joy': "not-stressed" , 'love': "not-stresse
                                    'sadness': "stressed", 'anger': "stressed", 'fear': "str
```

In [6]:

```
# check if pos and neg sentiments
df_reidx.label.value_counts()
```

Out[6]:

```
stressed        11598
not-stressed     8401
Name: label, dtype: int64
```

In [7]:

```
df_reidx['length'] = df_reidx['sentence'].apply(len) # number of characters
df_reidx['length'].describe() # info()
```

Out[7]:

```
count    19999.000000
mean        96.671784
std         55.778779
min          7.000000
25%         53.000000
50%         86.000000
75%        129.000000
max        300.000000
Name: length, dtype: float64
```

In [8]:

```
df_reidx.tail()
```

Out[8]:

	sentence	emotion	label	length
19994	im having ssa examination tomorrow in the morn...	sadness	stressed	191
19995	i constantly worry about their fight against n...	joy	not-stressed	173
19996	i feel its important to share this info for th...	joy	not-stressed	80
19997	i truly feel that if you are passionate enough...	joy	not-stressed	105
19998	i feel like i just wanna buy any cute make up ...	joy	not-stressed	74

Text Preprocessing

To clean the sentences,we do text preprocessing.

- Decontracted
- Data cleaning
- Spell check
- Lemmatization
- Nomalization

In [9]:

```
from tqdm import tqdm
import re
from bs4 import BeautifulSoup
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

def decontracted(phrase):
    """
    We first define a function to expand the contracted phrase into normal words
    """
    # specific
    phrase = re.sub(r"wont", "will not", phrase)
    phrase = re.sub(r"wouldnt", "would not", phrase)
    phrase = re.sub(r"shouldnt", "should not", phrase)
    phrase = re.sub(r"couldnt", "could not", phrase)
    phrase = re.sub(r"cudnt", "could not", phrase)
    phrase = re.sub(r"cant", "can not", phrase)
    phrase = re.sub(r"dont", "do not", phrase)
    phrase = re.sub(r"doesnt", "does not", phrase)
    phrase = re.sub(r"didnt", "did not", phrase)
    phrase = re.sub(r"wasnt", "was not", phrase)
    phrase = re.sub(r"werent", "were not", phrase)
    phrase = re.sub(r"havent", "have not", phrase)
    phrase = re.sub(r"hadnt", "had not", phrase)

    # general
    phrase = re.sub(r"n\ t", " not", phrase)
    #phrase = re.sub(r"\re", " are", phrase)
    phrase = re.sub(r"\ s ", " is ", phrase) # prime
    phrase = re.sub(r"\ d ", " would ", phrase)
    phrase = re.sub(r"\ ll ", " will ", phrase)
    phrase = re.sub(r"\dunno", "do not ", phrase)
    phrase = re.sub(r"ive ", "i have ", phrase)
    phrase = re.sub(r"im ", "i am ", phrase)
    phrase = re.sub(r"i m ", "i am ", phrase)
    phrase = re.sub(r" w ", " with ", phrase)

    return phrase


def clean_text(df):
    """
    Clean the review texts
    """
    cleaned_review = []

    for review_text in tqdm(df['sentence']):

        # expand the contracted words
        review_text = decontracted(review_text)
        #remove html tags
        review_text = BeautifulSoup(review_text, 'lxml').get_text().strip() # re.sub(r'<.*?

        #remove non-alphabetic characters
        review_text = re.sub("[^a-zA-Z]"," ", review_text)

        #remove url
        review_text = re.sub(r'https?://\S+|www\.\S+', '', review_text)
```

```
        #Removing punctutation, string.punctuation in python consists of !"#$%&\'()*+,-./:;
        review_text = review_text.translate(str.maketrans('', '', string.punctuation))
        # ''.join([char for char in movie_text_data if char not in string.punctuation])

        # remove emails
        review_text = re.sub(r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)", '', rev

        cleaned_review.append(review_text)

    return cleaned_review

df_reidx['cleaned_sentence'] = clean_text(df_reidx)
df_reidx.head()
```

```
100%|██████████| 19999/19999 [00:05<00:00, 3507.30it/s]
```

Out[9]:

	sentence	emotion	label	length	cleaned_sentence
0	i didnt feel humiliated	sadness	stressed	23	i did not feel humiliated
1	i can go from feeling so hopeless to so damned...	sadness	stressed	108	i can go from feeling so hopeless to so damned...
2	im grabbing a minute to post i feel greedy wrong	anger	stressed	48	i am grabbing a minute to post i feel greedy w...
3	i am ever feeling nostalgic about the fireplac...	love	not-stressed	92	i am ever feeling nostalgic about the fireplac...
4	i am feeling grouchy	anger	stressed	20	i am feeling grouchy

additional lemmatization

In [10]:

```
import nltk
nltk.download('punkt')
def remove_stopwords(phrase):
    remove_sw = []
    tokenizer = RegexpTokenizer(r'[a-zA-Z0-9]+')
    stop_words = stopwords.words('english')

    for review_text in tqdm(phrase):
        tokens = word_tokenize(review_text)
        tokens = [word for word in tokens if not word in stop_words]
        remove_sw.append(tokens)
    return remove_sw

df_reidx['cleaned_sentence'] = remove_stopwords(df_reidx['cleaned_sentence'])
df_reidx.head()
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno -3] Temporary
[nltk_data]     failure in name resolution>

100%|██████████| 19999/19999 [00:05<00:00, 3742.51it/s]
```

Out[10]:

	sentence	emotion	label	length	cleaned_sentence
0	i didnt feel humiliated	sadness	stressed	23	[feel, humiliated]
1	i can go from feeling so hopeless to so damned...	sadness	stressed	108	[go, feeling, hopeless, damned, hopeful, aroun...
2	im grabbing a minute to post i feel greedy wrong	anger	stressed	48	[grabbing, minute, post, feel, greedy, wrong]
3	i am ever feeling nostalgic about the fireplac...	love	not-stressed	92	[ever, feeling, nostalgic, fireplace, know, st...
4	i am feeling grouchy	anger	stressed	20	[feeling, grouchy]

In [11]:

```
#stemming for extract the actual meaning of the words
from nltk.stem import PorterStemmer

def stemming(phrase):
    stemmer = PorterStemmer()
    stem_output=[]
    stemmed=[]
    for review_text in tqdm(phrase):
        stemmed = [stemmer.stem(word) for word in review_text]
        stem_output.append(stemmed)
    return stem_output

df_reidx['cleaned_sentence'] = stemming(df_reidx['cleaned_sentence'])
df_reidx['cleaned_sentence'].head()
```

```
100%|██████████| 19999/19999 [00:05<00:00, 3420.42it/s]
```

Out[11]:

```
0                                      [feel, humili]
1    [go, feel, hopeless, damn, hope, around, someo...
2              [grab, minut, post, feel, greedi, wrong]
3    [ever, feel, nostalg, fireplac, know, still, n...
4                                     [feel, grouchi]
Name: cleaned_sentence, dtype: object
```

In [12]:

```
def to_sentence(phrase):
    sentence=[]
    for words in tqdm(phrase):
        sentence.append((" ").join(words))
    return sentence
df_reidx['cleaned_sentence']=to_sentence(df_reidx['cleaned_sentence'])
df_reidx['cleaned_sentence'].head()
```

```
100%|██████████| 19999/19999 [00:00<00:00, 740934.06it/s]
```

Out[12]:

```
0                                        feel humili
1    go feel hopeless damn hope around someon care ...
2                    grab minut post feel greedi wrong
3    ever feel nostalg fireplac know still noth pro...
4                                       feel grouchi
Name: cleaned_sentence, dtype: object
```

Feature Engineering

CounterVectorize: tokenization:

In [13]:

```
# convert the cleaned sentences to vectors
token = RegexpTokenizer(r'[a-zA-Z0-9]+')
# a built-in stop word list for english is used
# all values of n such than min_n<=n<= max_n will be used. (1,1): only unigrams, (1,2):uni&
# max_df: when building the vocabulary, ignore terms that have a document frequency strictl
# min_df: ignore terms that have a document frequency strictly lower than the given thresho

vectorizer = CountVectorizer(stop_words='english', max_df=0.5, min_df=3, ngram_range=(1,1),
x = vectorizer.fit_transform(df_reidx.cleaned_sentence)
y = df_reidx.label.values

print("X.shape : ",x.shape)
print("y.shape : ",y.shape)
```

```
X.shape :  (19999, 4374)
y.shape :  (19999,)
```

Train Test split

In [14]:

```
# do shuffle to make neg and pos data of data set split equaly in the test and training set
# do random_sate for making it settle when we run this code repeatedly
train_idx, test_idx = train_test_split(np.arange(df_reidx.shape[0]), test_size=0.3,shuffle=

x_train = x[train_idx]
y_train = y[train_idx]

x_test = x[test_idx]
y_test = y[test_idx]
print("Number of training examples:{}".format(len(train_idx)))
print("Number of testing examples:{}\n".format(len(test_idx)))
print("Training data: X_train : {}, y_train : {}".format(x_train.shape, y_train.shape))
print("Testing data: X_test : {}, y_test : {}".format(x_test.shape, y_test.shape))
```

```
Number of training examples:13999
Number of testing examples:6000

Training data: X_train : (13999, 4374), y_train : (13999,)
Testing data: X_test : (6000, 4374), y_test : (6000,)
```

In [15]:

```
x_train.shape
```

Out[15]:

```
(13999, 4374)
```

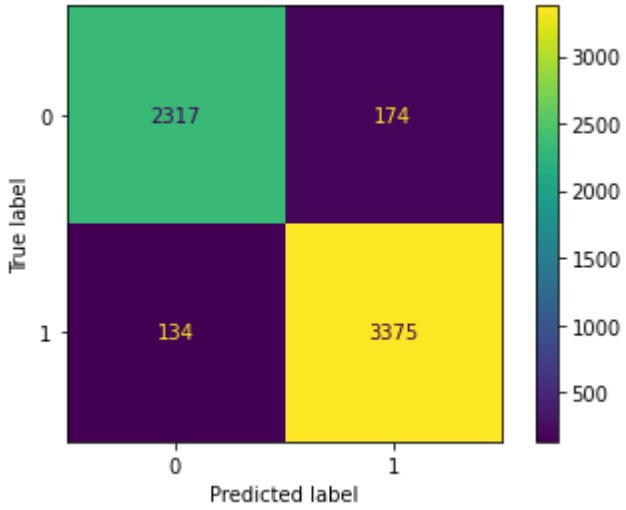
Model Training

Logistic Regression

In [16]:

```
# fit a logistic regression classifier on the training data use default settings
lr_clf = LogisticRegression()
lr_clf.fit(x_train, y_train)

# make prediction on testing data
y_pred_test_lr = lr_clf.predict(x_test)
y_predprob_lr = lr_clf.predict_proba(x_test)
matrix_lr = confusion_matrix(y_test,y_pred_test_lr)
print(classification_report(y_test, y_pred_test_lr))
print("\nAccuracy for Logistic Regression model:",metrics.accuracy_score(y_test, y_pred_tes
print("\n")
y_predict = lr_clf.predict(x_test)
matrix_display = ConfusionMatrixDisplay(matrix_lr).plot()
```

```
              precision    recall  f1-score   support

not-stressed       0.95      0.93      0.94      2491
    stressed       0.95      0.96      0.96      3509

    accuracy                           0.95      6000
   macro avg       0.95      0.95      0.95      6000
weighted avg       0.95      0.95      0.95      6000


Accuracy for Logistic Regression model: 0.9486666666666667
```



Naive Bayes classifier

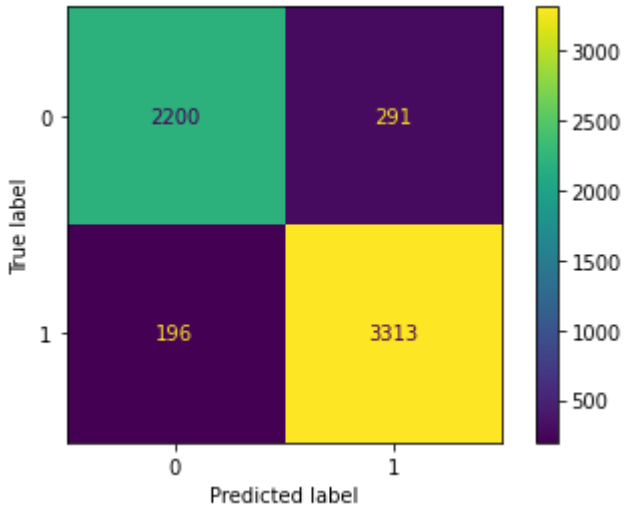
BernouliNB

A binary algorithm used when the feature is present or not.

In [17]:

```
from sklearn.naive_bayes import BernoulliNB
nb_clf = BernoulliNB()
nb_clf.fit(x_train, y_train)
# make prediction on testing data
y_pred_test_nb = nb_clf.predict(x_test)
y_predprob_nb = nb_clf.predict_proba(x_test)
matrix_nb = confusion_matrix(y_test,y_pred_test_nb)
print(classification_report(y_test, y_pred_test_nb))
print("\nAccuracy for Bernouli Naive Bayes model:",metrics.accuracy_score(y_test, y_pred_te
print("\n")
matrix_display = ConfusionMatrixDisplay(matrix_nb).plot()
```

```
              precision    recall  f1-score   support

not-stressed       0.92      0.88      0.90      2491
    stressed       0.92      0.94      0.93      3509

    accuracy                           0.92      6000
   macro avg       0.92      0.91      0.92      6000
weighted avg       0.92      0.92      0.92      6000


Accuracy for Bernouli Naive Bayes model: 0.9188333333333333
```



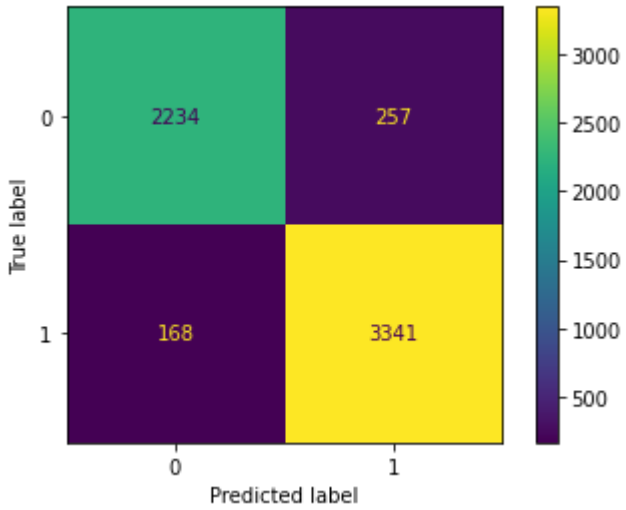
MultinominaliNB

It consider a feature vector where a given term represents the number of times it appears or very ofen, such as frequency.

In [18]:

```
mnb = MultinomialNB()
mnb.fit(x_train, y_train)
# make prediction on testing data
y_pred_test_mnb = mnb.predict(x_test)
y_predprob_mnb = mnb.predict_proba(x_test)
matrix = confusion_matrix(y_test,y_pred_test_mnb)
print(classification_report(y_test, y_pred_test_mnb))
print("\nAccuracy for multinominal Naive Bayes model:",metrics.accuracy_score(y_test, y_pre
print("\n")

y_predict = mnb.predict(x_test)
cm = confusion_matrix(y_test, y_predict)
cm_display = ConfusionMatrixDisplay(cm).plot()
```

```
              precision    recall  f1-score   support

not-stressed       0.93      0.90      0.91      2491
    stressed       0.93      0.95      0.94      3509

    accuracy                           0.93      6000
   macro avg       0.93      0.92      0.93      6000
weighted avg       0.93      0.93      0.93      6000


Accuracy for multinominal Naive Bayes model: 0.9291666666666667
```



Cross validataion

In [19]:

```
acc_score_lr = metrics.accuracy_score(y_pred_test_lr,y_test)
prec_score_lr = precision_score(y_test,y_pred_test_lr, average='macro')
recall_lr = recall_score(y_test, y_pred_test_lr,average='macro')
f1_lr = f1_score(y_test,y_pred_test_nb,average='macro')
matrix_lr = confusion_matrix(y_test,y_pred_test_lr)
print('Logistic Regression Model\n')
print(str('Accuracy: '+'{:04.2f}'.format(acc_score_lr*100))+'%')
print(str('Precision: '+'{:04.2f}'.format(prec_score_lr*100))+'%')
print(str('Recall: '+'{:04.2f}'.format(recall_lr*100))+'%')
print('F1 Score: ',f1_lr)
print(matrix_lr)
```

```
Logistic Regression Model

Accuracy: 94.87%
Precision: 94.82%
Recall: 94.60%
F1 Score:  0.9159408365025967
[[2317  174]
 [ 134 3375]]
```

BernouliNB

In [20]:

```
acc_score_nb = metrics.accuracy_score(y_pred_test_nb,y_test)
prec_score_nb = precision_score(y_test,y_pred_test_nb, average='macro')
recall_nb = recall_score(y_test, y_pred_test_nb,average='macro')
f1_nb = f1_score(y_test,y_pred_test_nb,average='macro')
matrix_nb = confusion_matrix(y_test,y_pred_test_nb)
print('Bernouli Naive Bayes Model\n')
print(str('Accuracy: '+'{:04.2f}'.format(acc_score_nb*100))+'%')
print(str('Precision: '+'{:04.2f}'.format(prec_score_nb*100))+'%')
print(str('Recall: '+'{:04.2f}'.format(recall_nb*100))+'%')
print('F1 Score: ',f1_nb)
print(matrix_nb)
```

```
Bernouli Naive Bayes Model

Accuracy: 91.88%
Precision: 91.87%
Recall: 91.37%
F1 Score:  0.9159408365025967
[[2200  291]
 [ 196 3313]]
```

MultinominaliNB

In [21]:

```
acc_score_mnb = metrics.accuracy_score(y_pred_test_mnb,y_test)
prec_score_mnb = precision_score(y_test,y_pred_test_mnb, average='macro')
recall_mnb = recall_score(y_test, y_pred_test_mnb,average='macro')
f1_mnb = f1_score(y_test,y_pred_test_mnb,average='macro')
matrix_mnb = confusion_matrix(y_test,y_pred_test_mnb)
print('Multimominal Naive Bayes Model\n')
print(str('Accuracy: '+'{:04.2f}'.format(acc_score_mnb*100))+'%')
print(str('Precision: '+'{:04.2f}'.format(prec_score_mnb*100))+'%')
print(str('Recall: '+'{:04.2f}'.format(recall_mnb*100))+'%')
print('F1 Score: ',f1_mnb)
print(matrix_mnb)
```

```
Multimominal Naive Bayes Model

Accuracy: 92.92%
Precision: 92.93%
Recall: 92.45%
F1 Score:  0.9266705125826067
[[2234  257]
 [ 168 3341]]
```

Explain the model prediction

Multimominal Naive Bayes Model has higher accuracy than Bernouli Naive Bayes Model.

In [22]:

```
test_data = df_reidx.iloc[test_idx]
test_data['pred_label'] = y_pred_test_lr
test_data.head(2)[['sentence','label','pred_label']]
# shows what the prediction label fit to the real label
```

Out[22]:

	sentence	label	pred_label
10650	i noticed several months ago that i d start fe...	stressed	stressed
2041	i love lots of different kinds of sports and l...	not-stressed	not-stressed

In [23]:

```
# shows what the prediction label does not fit to the real label
test_data[test_data['label'] != test_data['pred_label']].head()[['sentence','label','pred_l
```

Out[23]:

	sentence	label	pred_label
11072	being subject to unfair treatment in a working...	stressed	not-stressed
960	i will put my hand on his scar covered chest a...	not-stressed	stressed

Predicted features of logistic regression model

In [24]:

```
feature_to_coef = {word: float("%.3f" % coef) for word, coef in zip(vectorizer.get_feature_

print("Top positive features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=True)[:10]
```

Top positive features:

Out[24]:

```
[('agit', 3.413),
 ('intimid', 3.325),
 ('tortur', 3.279),
 ('reluct', 3.207),
 ('enviou', 3.165),
 ('punish', 3.134),
 ('groggi', 3.055),
 ('jealou', 2.976),
 ('weird', 2.972),
 ('gloomi', 2.922)]
```

In [25]:

```
# most of the words are reliable evidence of indicating negative sentiments
print("Top negative features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=False)[:10]
```

Top negative features:

Out[25]:

```
[('sympathet', -3.849),
 ('superior', -3.514),
 ('satisfi', -3.406),
 ('nostalg', -3.16),
 ('belov', -3.104),
 ('naughti', -3.097),
 ('resolv', -3.081),
 ('passion', -3.079),
 ('energet', -3.046),
 ('intellig', -3.037)]
```

Predicted features of BernouliNB

In [26]:

```
feature_to_coef = {word: float("%.3f" % coef) for word, coef in zip(vectorizer.get_feature_

print("Top positive features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=True)[:10]
```

Top positive features:

Out[26]:

```
[('like', -1.84),
 ('noth', -2.605),
 ('know', -2.835),
 ('realli', -2.919),
 ('time', -2.942),
 ('littl', -2.944),
 ('make', -3.05),
 ('want', -3.069),
 ('think', -3.138),
 ('thing', -3.158)]
```

In [27]:

```
# most of the words are reliable evidence of indicating negative sentiments
print("Top negative features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=False)[:10]
```

Top negative features:

Out[27]:

```
[('abyss', -8.999),
 ('accent', -8.999),
 ('accumul', -8.999),
 ('acquaint', -8.999),
 ('actio', -8.999),
 ('adrenalin', -8.999),
 ('advoc', -8.999),
 ('aesthet', -8.999),
 ('affection', -8.999),
 ('al', -8.999)]
```

Predicted features of multinomial NB

In [28]:

```
feature_to_coef = {word: float("%.3f" % coef) for word, coef in zip(vectorizer.get_feature_

print("Top positive features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=True)[:10]
```

Top positive features:

Out[28]:

```
[('like', -3.715),
 ('noth', -4.482),
 ('know', -4.748),
 ('realli', -4.819),
 ('littl', -4.86),
 ('time', -4.86),
 ('make', -4.949),
 ('want', -4.966),
 ('thing', -5.068),
 ('think', -5.071)]
```

In [29]:

```
# most of the words are reliable evidence of indicating negative sentiments
print("Top negative features:")
sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=False)[:10]
```

Top negative features:

Out[29]:

```
[('abyss', -10.965),
 ('accent', -10.965),
 ('accumul', -10.965),
 ('acquaint', -10.965),
 ('actio', -10.965),
 ('adrenalin', -10.965),
 ('advoc', -10.965),
 ('aesthet', -10.965),
 ('affection', -10.965),
 ('al', -10.965)]
```

Conclusion

In [30]:

```
text=['i am not feeling well', 'i want to make this project better', 'i feel aaaaaaah']
test_result = lr_clf.predict(vectorizer.transform(text))
print(test_result)
```

```
['stressed' 'not-stressed' 'stressed']
```