Natural Language Processing Template

Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [2]:
```
df = pd.read_csv("Restaurant_Reviews.tsv", sep="\t", quoting=3) # quoting=3 is remove "
df.head()
```

Out[2]:

	Review	Liked
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 [3]:
```
df.isnull().sum()
```

Out[3]:
```
Review    0
Liked     0
dtype: int64
```

In [4]:
```
# X = df["Review"]
# X.head()
```

In [5]:
```
# y = df["Liked"]
# y.head()
```

Cleaning the texts

In [6]:
```
# import nltk
# nltk.download('stopwords')
```

In [7]:
```
from nltk.corpus import stopwords

stopwords
```

Out[7]:
```
<WordListCorpusReader in 'C:\\Users\\Perry\\AppData\\Roaming\\nltk_data\\corpora\\stopwords'>
```

In [8]:
```
stopwords.words
```

Out[8]:
```
<bound method WordListCorpusReader.words of <WordListCorpusReader in 'C:\\Users\\Perry\\AppData\\Roaming\\nltk_data\\corpora
\\stopwords'>>
```

In [9]:
```
stopwords.words(fileids="english")
```

Out[9]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
```

In [10]:
```
%%time

import re
import nltk
# nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
from tqdm import tqdm

corpus = []
for row in tqdm(range(0, len(df))):
    review = re.sub(pattern="[^a-zA-Z]", repl=" ", string=df["Review"][row])
    review = review.lower()
    review = review.split()
    Porter = PorterStemmer()
    all_stopwords = stopwords.words("english")
    all_stopwords.remove("not") # remove "not" stopword
    review = [Porter.stem(word) for word in review if not word in all_stopwords]
    review = " ".join(review)
    corpus.append(review)
```

```
100%|██████████| 1000/1000 [00:00<00:00, 2119.70it/s]
Wall time: 484 ms
```

In [11]:
```
corpus
```

Out[11]:
```
['wow love place',
 'crust not good',
 'not tasti textur nasti',
 'stop late may bank holiday rick steve recommend love',
 'select menu great price',
 'get angri want damn pho',
 'honeslti tast fresh',
 'potato like rubber could tell made ahead time kept warmer',
 'fri great',
 'great touch',
 'servic prompt',
 'would not go back',
 'cashier care ever say still end wayyy overpr',
 'tri cape cod ravoli chicken cranberri mmmm',
 'disgust pretti sure human hair',
 'shock sign indic cash',
 'highli recommend',
 'waitress littl slow servic',
 'place not worth time let alon vega',
 'not like',
```

Creating the Bag of Words model

In [12]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray() # features must be 2d
y = df["Liked"]
```

In [13]:
```
# sparse matrix is 1000, 1500
X.shape
```

Out[13]: (1000, 1500)

In [14]:
```
len(X[0])
```

Out[14]: 1500

Splitting the dataset into the Training set and Test set

In [15]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [16]:
```
X_train.shape
```

Out[16]: (800, 1500)

In [17]:
```
X_test.shape
```

Out[17]: (200, 1500)

In [18]:
```
y_train.shape
```

Out[18]: (800,)

In [19]:
```
y_test.shape
```

Out[19]: (200,)

Using CountVectorizer

Decision Trees

CountVectorizer before train_test_split

In [20]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray() # Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

dtree_classifier = DecisionTreeClassifier(criterion="entropy", random_state=0)
dtree_classifier.fit(X_train, y_train)

dtree_predictions = dtree_classifier.predict(X_test)

print(confusion_matrix(y_test, dtree_predictions))
print(classification_report(y_test, dtree_predictions))
print(accuracy_score(y_test, dtree_predictions))
```

```
[[78 19]
 [31 72]]
              precision    recall  f1-score   support

           0       0.72      0.80      0.76        97
           1       0.79      0.70      0.74       103

    accuracy                           0.75       200
   macro avg       0.75      0.75      0.75       200
weighted avg       0.75      0.75      0.75       200
```

```
0.75
```

CountVectorizer after train_test_split

In [21]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

dtree_classifier = DecisionTreeClassifier(criterion="entropy", random_state=0)
dtree_classifier.fit(X_train, y_train)

dtree_predictions = dtree_classifier.predict(X_test)

print(confusion_matrix(y_test, dtree_predictions))
print(classification_report(y_test, dtree_predictions))
print(accuracy_score(y_test, dtree_predictions))
```

```
[[80 17]
 [39 64]]
              precision    recall  f1-score   support

           0       0.67      0.82      0.74        97
           1       0.79      0.62      0.70       103

    accuracy                           0.72       200
   macro avg       0.73      0.72      0.72       200
weighted avg       0.73      0.72      0.72       200

0.72
```

K Nearst Neighbors(No Feature Scaling)

CountVectorizer before train_test_split

In [22]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[68 29]
 [42 61]]
              precision    recall  f1-score   support

           0       0.62      0.70      0.66        97
           1       0.68      0.59      0.63       103

    accuracy                           0.65       200
   macro avg       0.65      0.65      0.64       200
weighted avg       0.65      0.65      0.64       200

0.645
```

CountVectorizer after train_test_split

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[74 23]
 [48 55]]
              precision    recall  f1-score   support

           0       0.61      0.76      0.68        97
           1       0.71      0.53      0.61       103

    accuracy                           0.65       200
   macro avg       0.66      0.65      0.64       200
weighted avg       0.66      0.65      0.64       200
```

0.645

K Nearst Neighbors(Feature Scaling)

CountVectorizer before train_test_split

In [24]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[65 32]
 [40 63]]
              precision    recall  f1-score   support

           0       0.62      0.67      0.64        97
           1       0.66      0.61      0.64       103

    accuracy                           0.64       200
   macro avg       0.64      0.64      0.64       200
weighted avg       0.64      0.64      0.64       200

0.64
```

CountVectorizer after train_test_split

In [25]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-25-6b1b9d4d4a57> in <module>
     13
     14 sc = StandardScaler()
---> 15 X_train = sc.fit_transform(X_train)
     16 X_test = sc.transform(X_test)
     17

~\anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transformation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transformation)

~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in fit(self, X, y, sample_weight)
    728         # Reset internal state before fitting
    729         self._reset()
--> 730         return self.partial_fit(X, y, sample_weight)
    731
    732     def partial_fit(self, X, y=None, sample_weight=None):

~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in partial_fit(self, X, y, sample_weight)
    791         if sparse.issparse(X):
    792             if self.with_mean:
--> 793                 raise ValueError(
    794                     "Cannot center sparse matrices: pass `with_mean=False` "
    795                     "instead. See docstring for motivation and alternatives.")

ValueError: Cannot center sparse matrices: pass `with_mean=False` instead. See docstring for motivation and alternatives.
```

Support Vector Machine(kernel="linear")

CountVectorizer before train_test_split

In [26]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

svc_classifier = SVC(kernel="linear", random_state=0)
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[79 18]
 [24 79]]
              precision    recall  f1-score   support

           0       0.77      0.81      0.79        97
           1       0.81      0.77      0.79       103

    accuracy                           0.79       200
   macro avg       0.79      0.79      0.79       200
weighted avg       0.79      0.79      0.79       200

0.79
```

CountVectorizer after train_test_split

In [27]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

svc_classifier = SVC(kernel="linear", random_state=0)
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[79 18]
 [22 81]]
              precision    recall  f1-score   support

           0       0.78      0.81      0.80        97
           1       0.82      0.79      0.80       103

    accuracy                           0.80       200
   macro avg       0.80      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200

0.8
```

Support Vector Machine(kernel="rbf")

CountVectorizer before train_test_split

In [28]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

svc_classifier = SVC(kernel="rbf")
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[89  8]
 [36 67]]
              precision    recall  f1-score   support

           0       0.71      0.92      0.80        97
           1       0.89      0.65      0.75       103

    accuracy                           0.78       200
   macro avg       0.80      0.78      0.78       200
weighted avg       0.81      0.78      0.78       200

0.78
```

CountVectorizer after train_test_split

In [29]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

svc_classifier = SVC(kernel="rbf")
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[87 10]
 [34 69]]
              precision    recall  f1-score   support

           0       0.72      0.90      0.80        97
           1       0.87      0.67      0.76       103

    accuracy                           0.78       200
   macro avg       0.80      0.78      0.78       200
weighted avg       0.80      0.78      0.78       200

0.78
```

Logistic Regression

CountVectorizer before train_test_split

In [30]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

lr_classifier = LogisticRegression(random_state=0)
lr_classifier.fit(X_train, y_train)

lr_predictions = lr_classifier.predict(X_test)

print(confusion_matrix(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
print(accuracy_score(y_test, lr_predictions))
```

```
[[80 17]
 [28 75]]
              precision    recall  f1-score   support

           0       0.74      0.82      0.78        97
           1       0.82      0.73      0.77       103

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.77       200
weighted avg       0.78      0.78      0.77       200

0.775
```

CountVectorizer after train_test_split

In [31]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

lr_classifier = LogisticRegression(random_state=0)
lr_classifier.fit(X_train, y_train)

lr_predictions = lr_classifier.predict(X_test)

print(confusion_matrix(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
print(accuracy_score(y_test, lr_predictions))
```

```
[[80 17]
 [28 75]]
              precision    recall  f1-score   support

           0       0.74      0.82      0.78        97
           1       0.82      0.73      0.77       103

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.77       200
weighted avg       0.78      0.78      0.77       200

0.775
```

Naive Bayes(GaussianNB)

CountVectorizer before train_test_split

In [32]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

gnb_classifier = GaussianNB()
gnb_classifier.fit(X_train, y_train)

gnb_predictions = gnb_classifier.predict(X_test)

print(confusion_matrix(y_test, gnb_predictions))
print(classification_report(y_test, gnb_predictions))
print(accuracy_score(y_test, gnb_predictions))
```

```
[[55 42]
 [12 91]]
              precision    recall  f1-score   support

           0       0.82      0.57      0.67        97
           1       0.68      0.88      0.77       103

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.72       200
weighted avg       0.75      0.73      0.72       200

0.73
```

CountVectorizer after train_test_split

In [33]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

# GaussianNB needs an array to fit the model
X_train = X_train.toarray()
X_test = X_test.toarray()

gnb_classifier = GaussianNB()
gnb_classifier.fit(X_train, y_train)

gnb_predictions = gnb_classifier.predict(X_test)

print(confusion_matrix(y_test, gnb_predictions))
print(classification_report(y_test, gnb_predictions))
print(accuracy_score(y_test, gnb_predictions))
```

```
[[55 42]
 [12 91]]
              precision    recall  f1-score   support

           0       0.82      0.57      0.67        97
           1       0.68      0.88      0.77       103

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.72       200
weighted avg       0.75      0.73      0.72       200

0.73
```

Naive Bayes(MultinomialNB)

CountVectorizer before train_test_split

In [34]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

mnb_classifier = MultinomialNB()
mnb_classifier.fit(X_train, y_train)

mnb_predictions = mnb_classifier.predict(X_test)

print(confusion_matrix(y_test, mnb_predictions))
print(classification_report(y_test, mnb_predictions))
print(accuracy_score(y_test, mnb_predictions))
```

```
[[74 23]
 [22 81]]
              precision    recall  f1-score   support

           0       0.77      0.76      0.77        97
           1       0.78      0.79      0.78       103

    accuracy                           0.78       200
   macro avg       0.77      0.77      0.77       200
weighted avg       0.77      0.78      0.77       200

0.775
```

CountVectorizer after train_test_split

In [35]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

mnb_classifier = MultinomialNB()
mnb_classifier.fit(X_train, y_train)

mnb_predictions = mnb_classifier.predict(X_test)

print(confusion_matrix(y_test, mnb_predictions))
print(classification_report(y_test, mnb_predictions))
print(accuracy_score(y_test, mnb_predictions))
```

```
[[77 20]
 [22 81]]
              precision    recall  f1-score   support

           0       0.78      0.79      0.79        97
           1       0.80      0.79      0.79       103

    accuracy                           0.79       200
   macro avg       0.79      0.79      0.79       200
weighted avg       0.79      0.79      0.79       200

0.79
```

Stochastic Gradient Descent (梯度下降法)

CountVectorizer before train_test_split

In [36]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

sgd_classifier = SGDClassifier()
sgd_classifier.fit(X_train, y_train)

sgd_predictions = sgd_classifier.predict(X_test)

print(confusion_matrix(y_test, sgd_predictions))
print(classification_report(y_test, sgd_predictions))
print(accuracy_score(y_test, sgd_predictions))
```

```
[[84 13]
 [45 58]]
              precision    recall  f1-score   support

           0       0.65      0.87      0.74        97
           1       0.82      0.56      0.67       103

    accuracy                           0.71       200
   macro avg       0.73      0.71      0.71       200
weighted avg       0.74      0.71      0.70       200

0.71
```

CountVectorizer after train_test_split

In [37]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

sgd_classifier = SGDClassifier()
sgd_classifier.fit(X_train, y_train)

sgd_predictions = sgd_classifier.predict(X_test)

print(confusion_matrix(y_test, sgd_predictions))
print(classification_report(y_test, sgd_predictions))
print(accuracy_score(y_test, sgd_predictions))
```

```
[[78 19]
 [27 76]]
              precision    recall  f1-score   support

           0       0.74      0.80      0.77        97
           1       0.80      0.74      0.77       103

    accuracy                           0.77       200
   macro avg       0.77      0.77      0.77       200
weighted avg       0.77      0.77      0.77       200

0.77
```

XGBoost

CountVectorizer before train_test_split

In [38]:
```
from sklearn.feature_extraction.text import CountVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)

xgb_predictions = xgb_classifier.predict(X_test)

print(confusion_matrix(y_test, xgb_predictions))
print(classification_report(y_test, xgb_predictions))
print(accuracy_score(y_test, xgb_predictions))
```

```
C:\Users\Perry\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier i
s deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_enc
oder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[13:26:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicit
ly set eval_metric if you'd like to restore the old behavior.
[[81 16]
 [40 63]]
              precision    recall  f1-score   support

           0       0.67      0.84      0.74        97
           1       0.80      0.61      0.69       103

    accuracy                           0.72       200
   macro avg       0.73      0.72      0.72       200
weighted avg       0.74      0.72      0.72       200

0.72
```

CountVectorizer after train_test_split

In [39]:
```
from sklearn.feature_extraction.text import CountVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)

xgb_predictions = xgb_classifier.predict(X_test)

print(confusion_matrix(y_test, xgb_predictions))
print(classification_report(y_test, xgb_predictions))
print(accuracy_score(y_test, xgb_predictions))
```

```
1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Exp
licitly set eval_metric if you'd like to restore the old behavior.

C:\Users\Perry\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifie
r is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_lab
el_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e.
0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[[79 18]
 [40 63]]
              precision    recall  f1-score   support

           0       0.66      0.81      0.73        97
           1       0.78      0.61      0.68       103

    accuracy                           0.71       200
   macro avg       0.72      0.71      0.71       200
weighted avg       0.72      0.71      0.71       200

0.71
```

Catboost (Tune by itself and no need to parameter tuning, good results when having multiple categorical variables)

CountVectorizer before train_test_split

In [40]:
```
from sklearn.feature_extraction.text import CountVectorizer
from catboost import CatBoostClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cat_classifier = CatBoostClassifier()
cat_classifier.fit(X_train, y_train)
```

```
cat_predictions = cat_classifier.predict(X_test)

print(confusion_matrix(y_test, cat_predictions))
print(classification_report(y_test, cat_predictions))
print(accuracy_score(y_test, cat_predictions))
```

```
992:    learn: 0.3814308        total: 7.86s    remaining: 55.4ms
993:    learn: 0.3813270        total: 7.87s    remaining: 47.5ms
994:    learn: 0.3810929        total: 7.87s    remaining: 39.6ms
995:    learn: 0.3808518        total: 7.88s    remaining: 31.7ms
996:    learn: 0.3804592        total: 7.89s    remaining: 23.7ms
997:    learn: 0.3803192        total: 7.9s     remaining: 15.8ms
998:    learn: 0.3801610        total: 7.9s     remaining: 7.91ms
999:    learn: 0.3800703        total: 7.91s    remaining: 0us
[[90  7]
 [41 62]]
              precision    recall  f1-score   support

           0       0.69      0.93      0.79        97
           1       0.90      0.60      0.72       103

    accuracy                           0.76       200
   macro avg       0.79      0.76      0.76       200
weighted avg       0.80      0.76      0.75       200

0.76
```

CountVectorizer after train_test_split (Dead Kernel)

In [41]:

```
# from sklearn.feature_extraction.text import CountVectorizer
# from catboost import CatBoostClassifier
# from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# X = pd.DataFrame(data=corpus, columns=["Review"])
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# cv = CountVectorizer(max_features=1500)

# X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transfo
# X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

# cat_classifier = CatBoostClassifier()
# cat_classifier.fit(X_train, y_train)

# cat_predictions = cat_classifier.predict(X_test)

# print(confusion_matrix(y_test, cat_predictions))
# print(classification_report(y_test, cat_predictions))
# print(accuracy_score(y_test, cat_predictions))
```

LightGBM (Light Gradient Boosting Machine)

CountVectorizer before train_test_split

In [42]:

```
from sklearn.feature_extraction.text import CountVectorizer
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

lgbm_classifier = LGBMClassifier()
lgbm_classifier.fit(X_train, y_train)

lgbm_predictions = lgbm_classifier.predict(X_test)

print(confusion_matrix(y_test, lgbm_predictions))
print(classification_report(y_test, lgbm_predictions))
print(accuracy_score(y_test, lgbm_predictions))
```

```
[[88  9]
 [54 49]]
              precision    recall  f1-score   support

           0       0.62      0.91      0.74        97
           1       0.84      0.48      0.61       103

    accuracy                           0.69       200
   macro avg       0.73      0.69      0.67       200
weighted avg       0.74      0.69      0.67       200

0.685
```

CountVectorizer after train_test_split

In [43]:

```
from sklearn.feature_extraction.text import CountVectorizer
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cv = CountVectorizer(max_features=1500)

X_train = cv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = cv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

lgbm_classifier = LGBMClassifier()
lgbm_classifier.fit(X_train, y_train)

lgbm_predictions = lgbm_classifier.predict(X_test)

print(confusion_matrix(y_test, lgbm_predictions))
```

```
print(classification_report(y_test, lgbm_predictions))
print(accuracy_score(y_test, lgbm_predictions))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-69abde74489d> in <module>
     12
     13 lgbm_classifier = LGBMClassifier()
---> 14 lgbm_classifier.fit(X_train, y_train)
     15
     16 lgbm_predictions = lgbm_classifier.predict(X_test)

~\anaconda3\lib\site-packages\lightgbm\sklearn.py in fit(self, X, y, sample_weight, init_score, eval_set, eval_names, eval_sample_weight, eval_class_weight, eval_init_score, eval_metric, early_stopping_rounds, verbose, feature_name, categorical_feature, callbacks, init_model)
    888                     valid_sets[i] = (valid_x, self._le.transform(valid_y))
    889
--> 890         super().fit(X, _y, sample_weight=sample_weight, init_score=init_score, eval_set=valid_sets,
    891                     eval_names=eval_names, eval_sample_weight=eval_sample_weight,
    892                     eval_class_weight=eval_class_weight, eval_init_score=eval_init_score,

~\anaconda3\lib\site-packages\lightgbm\sklearn.py in fit(self, X, y, sample_weight, init_score, group, eval_set, eval_names, eval_sample_weight, eval_class_weight, eval_init_score, eval_group, eval_metric, early_stopping_rounds, verbose, feature_name, categorical_feature, callbacks, init_model)
    681             init_model = init_model.booster_
    682
--> 683         self._Booster = train(params, train_set,
    684                               self.n_estimators, valid_sets=valid_sets, valid_names=eval_names,
    685                               early_stopping_rounds=early_stopping_rounds,

~\anaconda3\lib\site-packages\lightgbm\engine.py in train(params, train_set, num_boost_round, valid_sets, valid_names, fobj, feval, init_model, feature_name, categorical_feature, early_stopping_rounds, evals_result, verbose_eval, learning_rates, keep_training_booster, callbacks)
    226     # construct booster
    227     try:
--> 228         booster = Booster(params=params, train_set=train_set)
    229         if is_valid_contain_train:
    230             booster.set_train_data_name(train_data_name)

~\anaconda3\lib\site-packages\lightgbm\basic.py in __init__(self, params, train_set, model_file, model_str, silent)
   2227                 )
   2228             # construct booster object
-> 2229             train_set.construct()
   2230             # copy the parameters from train_set
   2231             params.update(train_set.get_params())

~\anaconda3\lib\site-packages\lightgbm\basic.py in construct(self)
   1466             else:
   1467                 # create train
-> 1468                 self._lazy_init(self.data, label=self.label,
   1469                                 weight=self.weight, group=self.group,
   1470                                 init_score=self.init_score, predictor=self._predictor,

~\anaconda3\lib\site-packages\lightgbm\basic.py in _lazy_init(self, data, label, reference, weight, group, init_score, predictor, silent, feature_name, categorical_feature, params)
   1264                 ctypes.byref(self.handle)))
   1265         elif isinstance(data, scipy.sparse.csr_matrix):
-> 1266             self.__init_from_csr(data, params_str, ref_dataset)
   1267         elif isinstance(data, scipy.sparse.csc_matrix):
   1268             self.__init_from_csc(data, params_str, ref_dataset)

~\anaconda3\lib\site-packages\lightgbm\basic.py in __init_from_csr(self, csr, params_str, ref_dataset)
   1374
   1375         ptr_indptr, type_ptr_indptr, __ = c_int_array(csr.indptr)
-> 1376         ptr_data, type_ptr_data, _ = c_float_array(csr.data)
   1377
   1378         assert csr.shape[1] <= MAX_INT32

~\anaconda3\lib\site-packages\lightgbm\basic.py in c_float_array(data)
    466             type_data = C_API_DTYPE_FLOAT64
    467         else:
--> 468             raise TypeError("Expected np.float32 or np.float64, met type({})"
    469                             .format(data.dtype))
    470     else:

TypeError: Expected np.float32 or np.float64, met type(int64)
```

Using TfidfVectorizer

Decision Trees

TfidfVectorizer before train_test_split

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

dtree_classifier = DecisionTreeClassifier(criterion="entropy", random_state=0)
dtree_classifier.fit(X_train, y_train)

dtree_predictions = dtree_classifier.predict(X_test)

print(confusion_matrix(y_test, dtree_predictions))
print(classification_report(y_test, dtree_predictions))
print(accuracy_score(y_test, dtree_predictions))
```

```
[[80 17]
 [39 64]]
              precision    recall  f1-score   support

           0       0.67      0.82      0.74        97
           1       0.79      0.62      0.70       103
```

```
    accuracy                           0.72       200
   macro avg       0.73      0.72      0.72       200
weighted avg       0.73      0.72      0.72       200

0.72
```

TfidfVectorizer after train_test_split

In [45]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

dtree_classifier = DecisionTreeClassifier(criterion="entropy", random_state=0)
dtree_classifier.fit(X_train, y_train)

dtree_predictions = dtree_classifier.predict(X_test)

print(confusion_matrix(y_test, dtree_predictions))
print(classification_report(y_test, dtree_predictions))
print(accuracy_score(y_test, dtree_predictions))
```

```
[[81 16]
 [37 66]]
              precision    recall  f1-score   support

           0       0.69      0.84      0.75        97
           1       0.80      0.64      0.71       103

    accuracy                           0.73       200
   macro avg       0.75      0.74      0.73       200
weighted avg       0.75      0.73      0.73       200

0.735
```

K Nearst Neighbors(No Feature Scaling)

TfidfVectorizer before train_test_split

In [46]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[79 18]
 [41 62]]
              precision    recall  f1-score   support

           0       0.66      0.81      0.73        97
           1       0.78      0.60      0.68       103

    accuracy                           0.70       200
   macro avg       0.72      0.71      0.70       200
weighted avg       0.72      0.70      0.70       200

0.705
```

TfidfVectorizer after train_test_split

In [47]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[80 17]
 [41 62]]
              precision    recall  f1-score   support

           0       0.66      0.82      0.73        97
           1       0.78      0.60      0.68       103
```

```
    accuracy                           0.71       200
   macro avg       0.72      0.71      0.71       200
weighted avg       0.72      0.71      0.71       200

0.71
```

K Nearst Neighbors(Feature Scaling)

TfidfVectorizer before train_test_split

In [48]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
[[84 13]
 [57 46]]
              precision    recall  f1-score   support

           0       0.60      0.87      0.71        97
           1       0.78      0.45      0.57       103

    accuracy                           0.65       200
   macro avg       0.69      0.66      0.64       200
weighted avg       0.69      0.65      0.63       200

0.65
```

TfidfVectorizer after train_test_split

In [49]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_classifier.fit(X_train, y_train)

knn_predictions = knn_classifier.predict(X_test)

print(confusion_matrix(y_test, knn_predictions))
print(classification_report(y_test, knn_predictions))
print(accuracy_score(y_test, knn_predictions))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-49-612390378f5a> in <module>
     13
     14 sc = StandardScaler()
---> 15 X_train = sc.fit_transform(X_train)
     16 X_test = sc.transform(X_test)
     17

~\anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transformation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transformation)

~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in fit(self, X, y, sample_weight)
    728         # Reset internal state before fitting
    729         self._reset()
--> 730         return self.partial_fit(X, y, sample_weight)
    731
    732     def partial_fit(self, X, y=None, sample_weight=None):

~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in partial_fit(self, X, y, sample_weight)
    791         if sparse.issparse(X):
    792             if self.with_mean:
--> 793                 raise ValueError(
    794                     "Cannot center sparse matrices: pass `with_mean=False` "
    795                     "instead. See docstring for motivation and alternatives.")

ValueError: Cannot center sparse matrices: pass `with_mean=False` instead. See docstring for motivation and alternatives.
```

Support Vector Machine(kernel="linear")

TfidfVectorizer before train_test_split

In [50]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

svc_classifier = SVC(kernel="linear", random_state=0)
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[84 13]
 [32 71]]
              precision    recall  f1-score   support

           0       0.72      0.87      0.79        97
           1       0.85      0.69      0.76       103

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.77       200
weighted avg       0.79      0.78      0.77       200

0.775
```

TfidfVectorizer after train_test_split

In [51]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

svc_classifier = SVC(kernel="linear", random_state=0)
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[83 14]
 [30 73]]
              precision    recall  f1-score   support

           0       0.73      0.86      0.79        97
           1       0.84      0.71      0.77       103

    accuracy                           0.78       200
   macro avg       0.79      0.78      0.78       200
weighted avg       0.79      0.78      0.78       200

0.78
```

Support Vector Machine(kernel="rbf")

TfidfVectorizer before train_test_split

In [52]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

svc_classifier = SVC(kernel="rbf")
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[87 10]
 [31 72]]
              precision    recall  f1-score   support

           0       0.74      0.90      0.81        97
           1       0.88      0.70      0.78       103

    accuracy                           0.80       200
   macro avg       0.81      0.80      0.79       200
weighted avg       0.81      0.80      0.79       200

0.795
```

TfidfVectorizer after train_test_split

In [53]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

svc_classifier = SVC(kernel="rbf")
svc_classifier.fit(X_train, y_train)

svc_predictions = svc_classifier.predict(X_test)

print(confusion_matrix(y_test, svc_predictions))
print(classification_report(y_test, svc_predictions))
print(accuracy_score(y_test, svc_predictions))
```

```
[[86 11]
 [29 74]]
              precision    recall  f1-score   support

           0       0.75      0.89      0.81        97
           1       0.87      0.72      0.79       103

    accuracy                           0.80       200
   macro avg       0.81      0.80      0.80       200
weighted avg       0.81      0.80      0.80       200

0.8
```

Logistic Regression

TfidfVectorizer before train_test_split

In [54]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

lr_classifier = LogisticRegression(random_state=0)
lr_classifier.fit(X_train, y_train)

lr_predictions = lr_classifier.predict(X_test)

print(confusion_matrix(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
print(accuracy_score(y_test, lr_predictions))
```

```
[[85 12]
 [32 71]]
              precision    recall  f1-score   support

           0       0.73      0.88      0.79        97
           1       0.86      0.69      0.76       103

    accuracy                           0.78       200
   macro avg       0.79      0.78      0.78       200
weighted avg       0.79      0.78      0.78       200

0.78
```

TfidfVectorizer after train_test_split

In [55]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

lr_classifier = LogisticRegression(random_state=0)
lr_classifier.fit(X_train, y_train)

lr_predictions = lr_classifier.predict(X_test)

print(confusion_matrix(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
print(accuracy_score(y_test, lr_predictions))
```

```
[[85 12]
 [28 75]]
              precision    recall  f1-score   support

           0       0.75      0.88      0.81        97
           1       0.86      0.73      0.79       103

    accuracy                           0.80       200
   macro avg       0.81      0.80      0.80       200
weighted avg       0.81      0.80      0.80       200

0.8
```

Naive Bayes(GaussianNB)

TfidfVectorizer before train_test_split

In [56]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

gnb_classifier = GaussianNB()
gnb_classifier.fit(X_train, y_train)

gnb_predictions = gnb_classifier.predict(X_test)

print(confusion_matrix(y_test, gnb_predictions))
print(classification_report(y_test, gnb_predictions))
print(accuracy_score(y_test, gnb_predictions))
```

```
[[57 40]
 [16 87]]
              precision    recall  f1-score   support

           0       0.78      0.59      0.67        97
           1       0.69      0.84      0.76       103

    accuracy                           0.72       200
   macro avg       0.73      0.72      0.71       200
weighted avg       0.73      0.72      0.71       200

0.72
```

TfidfVectorizer after train_test_split

In [57]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

# GaussianNB needs an array to fit the model
X_train = X_train.toarray()
X_test = X_test.toarray()

gnb_classifier = GaussianNB()
gnb_classifier.fit(X_train, y_train)

gnb_predictions = gnb_classifier.predict(X_test)

print(confusion_matrix(y_test, gnb_predictions))
print(classification_report(y_test, gnb_predictions))
print(accuracy_score(y_test, gnb_predictions))
```

```
[[58 39]
 [14 89]]
              precision    recall  f1-score   support

           0       0.81      0.60      0.69        97
           1       0.70      0.86      0.77       103

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.73       200
weighted avg       0.75      0.73      0.73       200

0.735
```

Naive Bayes(MultinomialNB)

TfidfVectorizer before train_test_split

In [58]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

mnb_classifier = MultinomialNB()
mnb_classifier.fit(X_train, y_train)

mnb_predictions = mnb_classifier.predict(X_test)

print(confusion_matrix(y_test, mnb_predictions))
print(classification_report(y_test, mnb_predictions))
print(accuracy_score(y_test, mnb_predictions))
```

```
[[75 22]
 [20 83]]
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        97
           1       0.79      0.81      0.80       103

    accuracy                           0.79       200
```

```
   macro avg       0.79      0.79      0.79       200
weighted avg       0.79      0.79      0.79       200

0.79
```

TfidfVectorizer after train_test_split

In [59]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

mnb_classifier = MultinomialNB()
mnb_classifier.fit(X_train, y_train)

mnb_predictions = mnb_classifier.predict(X_test)

print(confusion_matrix(y_test, mnb_predictions))
print(classification_report(y_test, mnb_predictions))
print(accuracy_score(y_test, mnb_predictions))
```

```
[[76 21]
 [21 82]]
              precision    recall  f1-score   support

           0       0.78      0.78      0.78        97
           1       0.80      0.80      0.80       103

    accuracy                           0.79       200
   macro avg       0.79      0.79      0.79       200
weighted avg       0.79      0.79      0.79       200

0.79
```

Stochastic Gradient Descent (梯度下降法)

TfidfVectorizer before train_test_split

In [60]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

sgd_classifier = SGDClassifier()
sgd_classifier.fit(X_train, y_train)

sgd_predictions = sgd_classifier.predict(X_test)

print(confusion_matrix(y_test, sgd_predictions))
print(classification_report(y_test, sgd_predictions))
print(accuracy_score(y_test, sgd_predictions))
```

```
[[81 16]
 [27 76]]
              precision    recall  f1-score   support

           0       0.75      0.84      0.79        97
           1       0.83      0.74      0.78       103

    accuracy                           0.79       200
   macro avg       0.79      0.79      0.78       200
weighted avg       0.79      0.79      0.78       200

0.785
```

TfidfVectorizer after train_test_split

In [61]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

sgd_classifier = SGDClassifier()
sgd_classifier.fit(X_train, y_train)

sgd_predictions = sgd_classifier.predict(X_test)

print(confusion_matrix(y_test, sgd_predictions))
print(classification_report(y_test, sgd_predictions))
print(accuracy_score(y_test, sgd_predictions))
```

```
[[76 21]
 [24 79]]
              precision    recall  f1-score   support

           0       0.76      0.78      0.77        97
           1       0.79      0.77      0.78       103

    accuracy                           0.78       200
```

```
   macro avg       0.78      0.78      0.77       200
weighted avg       0.78      0.78      0.78       200

0.775
```

XGBoost

TfidfVectorizer before train_test_split

In [62]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)

xgb_predictions = xgb_classifier.predict(X_test)

print(confusion_matrix(y_test, xgb_predictions))
print(classification_report(y_test, xgb_predictions))
print(accuracy_score(y_test, xgb_predictions))
```

```
[13:27:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
C:\Users\Perry\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[[87 10]
 [44 59]]
              precision    recall  f1-score   support

           0       0.66      0.90      0.76        97
           1       0.86      0.57      0.69       103

    accuracy                           0.73       200
   macro avg       0.76      0.73      0.72       200
weighted avg       0.76      0.73      0.72       200

0.73
```

TfidfVectorizer after train_test_split

In [63]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)

xgb_predictions = xgb_classifier.predict(X_test)

print(confusion_matrix(y_test, xgb_predictions))
print(classification_report(y_test, xgb_predictions))
print(accuracy_score(y_test, xgb_predictions))
```

```
[13:27:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[[81 16]
 [42 61]]
              precision    recall  f1-score   support

           0       0.66      0.84      0.74        97
           1       0.79      0.59      0.68       103

    accuracy                           0.71       200
   macro avg       0.73      0.71      0.71       200
weighted avg       0.73      0.71      0.71       200

0.71
```

```
C:\Users\Perry\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Catboost (Tune by itself and no need to parameter tuning, good results when having multiple categorical variables)

TfidfVectorizer before train_test_split

In [64]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from catboost import CatBoostClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]
```

```
y = df['Liked']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

cat_classifier = CatBoostClassifier()
cat_classifier.fit(X_train, y_train)

cat_predictions = cat_classifier.predict(X_test)

print(confusion_matrix(y_test, cat_predictions))
print(classification_report(y_test, cat_predictions))
print(accuracy_score(y_test, cat_predictions))
```

```
993:    learn: 0.3691494        total: 7.09s    remaining: 42.8ms
994:    learn: 0.3690334        total: 7.09s    remaining: 35.7ms
995:    learn: 0.3687984        total: 7.1s     remaining: 28.5ms
996:    learn: 0.3684584        total: 7.11s    remaining: 21.4ms
997:    learn: 0.3682816        total: 7.12s    remaining: 14.3ms
998:    learn: 0.3680785        total: 7.12s    remaining: 7.13ms
999:    learn: 0.3676815        total: 7.13s    remaining: 0us

[[90  7]
 [44 59]]
              precision    recall  f1-score   support

           0       0.67      0.93      0.78        97
           1       0.89      0.57      0.70       103

    accuracy                           0.74       200
   macro avg       0.78      0.75      0.74       200
weighted avg       0.79      0.74      0.74       200

0.745
```

TfidfVectorizer after train_test_split (Dead Kernel)

In [65]:

```
# from sklearn.feature_extraction.text import TfidfVectorizer
# from catboost import CatBoostClassifier
# from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# X = pd.DataFrame(data=corpus, columns=["Review"])
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# tv = TfidfVectorizer(max_features=1500)

# X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transfor
# X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

# cat_classifier = CatBoostClassifier()
# cat_classifier.fit(X_train, y_train)

# cat_predictions = cat_classifier.predict(X_test)

# print(confusion_matrix(y_test, cat_predictions))
# print(classification_report(y_test, cat_predictions))
# print(accuracy_score(y_test, cat_predictions))
```

LightGBM (Light Gradient Boosting Machine)

TfidfVectorizer before train_test_split

In [66]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

tv = TfidfVectorizer(max_features=1500)
X = tv.fit_transform(corpus).toarray()# Need a list inside of fit,fit_transform,transform
y = df["Liked"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

lgbm_classifier = LGBMClassifier()
lgbm_classifier.fit(X_train, y_train)

lgbm_predictions = lgbm_classifier.predict(X_test)

print(confusion_matrix(y_test, lgbm_predictions))
print(classification_report(y_test, lgbm_predictions))
print(accuracy_score(y_test, lgbm_predictions))
```

```
[[88  9]
 [56 47]]
              precision    recall  f1-score   support

           0       0.61      0.91      0.73        97
           1       0.84      0.46      0.59       103

    accuracy                           0.68       200
   macro avg       0.73      0.68      0.66       200
weighted avg       0.73      0.68      0.66       200

0.675
```

TfidfVectorizer after train_test_split

In [67]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

X = pd.DataFrame(data=corpus, columns=["Review"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tv = TfidfVectorizer(max_features=1500)

X_train = tv.fit_transform(raw_documents=X_train["Review"].to_list())# Need a list inside of fit,fit_transform,transform
X_test = tv.transform(raw_documents=X_test["Review"].to_list())# Need a list inside of fit,fit_transform,transform

lgbm_classifier = LGBMClassifier()
```

```
lgbm_classifier.fit(X_train, y_train)

lgbm_predictions = lgbm_classifier.predict(X_test)

print(confusion_matrix(y_test, lgbm_predictions))
print(classification_report(y_test, lgbm_predictions))
print(accuracy_score(y_test, lgbm_predictions))
```

```
[[84 13]
 [51 52]]
              precision    recall  f1-score   support

           0       0.62      0.87      0.72        97
           1       0.80      0.50      0.62       103

    accuracy                           0.68       200
   macro avg       0.71      0.69      0.67       200
weighted avg       0.71      0.68      0.67       200

0.68
```