We will use the test data set of rotten tomato review
https://www.kaggle.com/datasets/ulrikthygepedersen/rotten-tomatoes-reviews

In machine learning algorithms, naïve bayes classification is a straight forward and powerful algorithm for classification task. Here I implementing the naïve bayes classification using python. Here, I build an naïve bayes classifier to predicting whether the review is rotten or fresh.

Naïve Bayes Algorithm:

It is a classification technique based on bayes theorem with an independence assumption among predictors. Naïve Bayes classification is based on applying Bayes theorem with strong independence assumption between the features. Naïve Bayes classification produces good results when we use it for textual data analysis such as Natural Language Processing.Naïve Bayes classifier applies the Bayes' theorem in practice. This classifier brings the power of Bayes' theorem to machine learning.

Naïve Bayes algorithm intuition:

Naïve bayes classifier uses the bayes theorem to predict the membership probabilities of each class such as probability of given data belongs to a particular class. The class with highest probability will considered as most likely class.

Here, naïve bayes classifier assumes that all the features are unrelated to each other. Presence or absence of a feature will not effect or influence the presence or absence of any another feature.

In real world datasets, we test a hypothesis given multiple evidence on features. So, the calculations become quite complicated.

Types of Naïve Bayes Algorithm:

Gaussian Naïve Bayes algorithm:

When we have continuous attribute values, we made an assumption that the values associated with each class are distributed according to Gaussian or Normal distribution

Multinomial Naïve Bayes algorithm:

Multinomial Naïve Bayes model, samples represent the frequencies with which certain events have been generated by a multinomial where pi is the probability that event i occurs. Multinomial Naïve Bayes algorithm is preferred to use on data that is multinomially distributed. It is used in text categorization classification.

Bernoulli Naïve Bayes algorithm:

This model is also popular for document classification tasks where binary term occurrence features are used rather than term frequencies.

The applications of the naïve bayes algorithm is the text classification and sentiment analysis.

Import libraries:

The first step will be importing the libraries which are the basic libraries such as numpy, pandas and re. These are the most important things for any machine learning model.

```
import numpy as np
import pandas as pd
import re
import string
```

Import data set:

We need to import the data set using the read_csv function. where it will helps us to load the csv files.

```
data=pd.read_csv("Documents/rt_reviews.csv",encoding="cp1252")
```

Exploratory data analysis:

To better understand of the data. We use some functions such as info(),value_counts(), isnull() and head(). Which will helps to understand the dataset in a better way.

```
data.head()
```

	Freshness	Review
0	fresh	Manakamana doesn't answer any questions, yet ...
1	fresh	Wilfully offensive and powered by a chest-thu...
2	rotten	It would be difficult to imagine material mor...
3	rotten	Despite the gusto its star brings to the role...
4	rotten	If there was a good idea at the core of this ...

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480000 entries, 0 to 479999
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   Freshness  480000 non-null  object
 1   Review     480000 non-null  object
dtypes: object(2)
memory usage: 7.3+ MB
```

```
data["Freshness"].value_counts()
```

```
fresh     240000
rotten    240000
Name: Freshness, dtype: int64
```

```
data.isnull().sum()
```

```
Freshness    0
Review       0
dtype: int64
```

Encoding:

In this stage we are going to remove the unwanted data. It is a process of removing the commoner morphological and inflexional endings from words in English. We need to perform this before we are going to build a model. Here we remove the stop_words, other than alphabets and numbers to perform a model.

```
a=[]
stop_words = ["a", "an", "the", "and", "or", "but", "is", "am", "are", "was", "were", "be", "being", "been", "have", "has", "had'
for i in data["Review"]:
        msg1=" "
        message = i.lower()
        message = message.split()
        words=[]
        for word in message:
            word=re.sub("[^a-zA-Z0-9/s]","",word)
            if word not in stop_words:
              words.append(word)

        msg1=" ".join(words)
        a.append(msg1)
```

```
data["Review"]=a
```

```
data["Review"] = data["Review"].apply(lambda x: x.translate(str.maketrans("", "", string.punctuation)))
```

Splitting data into train, development and test

Merge the dataset into one. And divide the dataset as train, development and test.

```
#we need to divide data into three catogoeries
size = data.shape[0]

train_size = int(size*0.6)
dev_size = int(size*0.2)
train_data = data.iloc[:train_size,:]
dev_data   = data.iloc[train_size:train_size+dev_size+1,:]
test_data  = data.iloc[train_size+dev_size:,:]
```

```
train_data.head()
```

	Freshness	Review
0	fresh	manakamana doesnt answer any questions yet mak...
1	fresh	wilfully offensive powered by chestthumping ma...
2	rotten	it difficult to imagine material more wrong fo...
3	rotten	despite gusto its star brings to role its hard...
4	rotten	if there good idea at core of this film its bu...

```
dev_data.head()
```

	Freshness	Review
288000	rotten	tepid attempt at making alien abduction yarn i...
288001	fresh	perfectly serviceable raceagainsttheclock thri...
288002	fresh	poetic pulp movie made with joy
288003	fresh	thirst just best horror movie of year so far p...
288004	fresh	expect same easy chemistry between two main le...

```
test_data.head()
```

	Freshness	Review
384000	rotten	film that abuses its ridiculous premise never ...
384001	fresh	for any age its amiable harmlessly entertainin...
384002	fresh	its incredibly stirring documentary one which ...
384003	fresh	lennons spirit like his music shines through t...
384004	rotten	try as it to build sense of mystery out of sto...

Build a vocabulary as list. Where i omitted rare words for example if the occurrence is less than five times. Then I dropped that words and also performed the reverse index as the key value.

```
# def count_of_word(data):
count_of_word={}
for i in train_data["Review"]:
    for word in i.split():
        if word in count_of_word:
            count_of_word[word]+=1
        else:
            count_of_word[word]=1
words=[i for i,j in count_of_word.items() if j>=5]
words[:10]
```

```
['doesnt',
 'answer',
 'any',
 'questions',
 'yet',
 'makes',
 'its',
 'point',
 'like',
 'rest']
```

```
#index to word conversion
t=0
word_to_index={}
index_to_word={}
for i in words:
    word_to_index[t]=i
    index_to_word[i]=t
    t+=1
```

```
word_to_index
```

```
{'doesnt': 0,
 'answer': 1,
 'any': 2,
 'questions': 3,
 'yet': 4,
 'makes': 5,
 'its': 6,
 'point': 7,
 'like': 8,
 'rest': 9,
 'of': 10,
 'our': 11,
 'planet': 12,
 'picturesque': 13,
 'far': 14,
 'from': 15,
 'kingdom': 16,
 'wilfully': 17,
 'offensive': 18,
```

```
index_to_wor
```

```
{0: 'doesnt',
 1: 'answer',
 2: 'any',
 3: 'questions',
 4: 'yet',
 5: 'makes',
 6: 'its',
 7: 'point',
 8: 'like',
 9: 'rest',
 10: 'of',
 11: 'our',
```

Calculating the probablility of occurrence of each word

```
#probability of occurrence of each word
prob_of_occurrence={}
for i in words:
    prob_of_occurrence[i]=count_of_word[i]/len(train_data["Freshness"])
```

```
prob_of_occurrence
```

```
 'point': 0.009607638888888889,
 'like': 0.0732326388888889,
 'rest': 0.0036006944444444446,
 'of': 0.6430034722222222,
 'our': 0.011840277777777778,
 'planet': 0.0012847222222222223,
 'picturesque': 0.00018402777777777778,
 'far': 0.014479166666666666,
 'from': 0.07544444444444444,
 'kingdom': 0.0007673611111111111,
 'wilfully': 7.638888888888889e-05,
 'offensive': 0.0013125,
 'powered': 0.00023958333333333332,
 'by': 0.07953125,
 'chestthumping': 2.777777777777778e-05,
 'machismo': 0.00019444444444444443,
 'good': 0.038322916666666665,
 'clean': 0.0006493055555555555,
 'fun': 0.02153125,
```

Calculating the Conditional probability based on the sentiment

```
#conditional probability
word_count_positive= np.zeros(len(words))
word_count_negative= np.zeros(len(words))
for i,j in zip(train_data["Freshness"],train_data["Review"]):
    for word in j.split():
#         print(word)
        if word in index_to_word:
            index=index_to_word[word]
            if(i=="fresh"):
                word_count_positive[index]+=1
            else:
                word_count_negative[index]+=1
```

```
word_count_positive
```

```
array([3.119e+03, 1.790e+02, 2.784e+03, ..., 2.000e+00, 5.000e+00,
       2.000e+00])
```

```
word_count_negative
```

```
array([5.321e+03, 2.190e+02, 4.546e+03, ..., 3.000e+00, 1.000e+00,
       3.000e+00])
```

```
#conditional probobility
count_positive=0
count_negative=0
for i in train_data["Freshness"]:
    if(i=="fresh"):
        count_positive+=1
    else:
        count_negative+=1
prob_word_count_positive=word_count_positive/count_positive
prob_word_count_negative=word_count_negative/count_negative
```

```
prob_word_count_positive
```

```
array([2.16340318e-02, 1.24158118e-03, 1.93104022e-02, ...,
       1.38724154e-05, 3.46810385e-05, 1.38724154e-05])
```

```
prob_word_count_negative
```

```
array([3.69953208e-02, 1.52264147e-03, 3.16069777e-02, ...,
       2.08581023e-05, 6.95270078e-06, 2.08581023e-05])
```

Comparing the effect of smoothing on the Conditional probability based on the sentiment.

```
alpha=1
prob_word_count_positive_smoothing=(word_count_positive+alpha)/(count_positive+alpha*len(words))
prob_word_count_negative_smoothing=(word_count_negative+alpha)/(count_negative+alpha*len(words))
```

```
prob_word_count_positive_smoothing
```

```
array([1.72983522e-02, 9.97981859e-04, 1.54409971e-02, ...,
       1.66330310e-05, 3.32660620e-05, 1.66330310e-05])
```

```
prob_word_count_negative_smoothing
```

```
array([2.95630534e-02, 1.22207286e-03, 2.52580240e-02, ...,
       2.22195065e-05, 1.11097533e-05, 2.22195065e-05])
```

```python
prob_positive = count_positive/len(train_data)
prob_positive
```

```
0.50059375
```

```python
prob_negative = count_negative/len(train_data)
prob_negative
```

```
0.49940625
```

```python
for i in range(10):
    word = words[i]
    print("\n"+word+':')
    print("\tP(word|fresh):", prob_word_count_positive_smoothing[i])
    print("\tP(word|rotten):", prob_word_count_negative_smoothing[i])
```

```
doesnt:
	P(word|fresh): 0.01729835222106407
	P(word|rotten): 0.029563053404583883

answer:
	P(word|fresh): 0.0009979818589075425
	P(word|rotten): 0.001222072857761829

any:
	P(word|fresh): 0.01544099709476392
	P(word|rotten): 0.02525802401928653

questions:
	P(word|fresh): 0.003265618416091903
	P(word|rotten): 0.0019108775594094056

yet:
	P(word|fresh): 0.014698055044243862
	P(word|rotten): 0.01088755818733266

makes:
	P(word|fresh): 0.023097735690049012
	P(word|rotten): 0.015892502027529967

its:
	P(word|fresh): 0.22021578585527046
	P(word|rotten): 0.2111075313017298

point:
	P(word|fresh): 0.005378013350779535
	P(word|rotten): 0.009993223050516049

like:
	P(word|fresh): 0.04798075003881041
	P(word|rotten): 0.06909711035317906

rest:
	P(word|fresh): 0.002045862810760462
	P(word|rotten): 0.003721767339547389
```

Calculating the accuracy using development dataset

```
# Development Data
def classifying(data,smoothing=True):
        v=[]
        for i in data["Review"]:
            msg1=" "
            message = i.lower()

            if smoothing:
                new_prob_fresh = prob_positive*len(word_count_positive)
                new_prob_rotten = prob_negative*len(word_count_negative)
            else:
                new_prob_fresh = prob_positive
                new_prob_rotten = prob_negative
            for i in message.split():
#                print(i)
                if i in index_to_word.keys():
#                    print(i)
                    index = index_to_word[i]
#                    print(index)
                    if smoothing:
                        new_prob_fresh*=prob_word_count_positive[index]+1/count_positive
                        new_prob_rotten*=prob_word_count_negative[index]+1/count_negative
                    else:
                        new_prob_fresh*=prob_word_count_positive[index]
                        new_prob_rotten*=prob_word_count_negative[index]
            if new_prob_fresh>new_prob_rotten:
                v.append("fresh")
            else:
                v.append("rotten")

        return v
```

```
a=classifying(dev_data)
```

```
dev_data["smoothing_prediction"] = a
```

Comparing the effect of smoothing.where with the effect of smoothing we got the accuracy 0f 79.59% and without smoothing we got the accuracy 79.47% .

```
dev_data
```

	Freshness	Review	smoothing_prediction
288000	rotten	tepid attempt at making alien abduction yarn i...	rotten
288001	fresh	perfectly serviceable raceagainsttheclock thri...	fresh
288002	fresh	poetic pulp movie made with joy	fresh
288003	fresh	thirst just best horror movie of year so far p...	fresh
288004	fresh	expect same easy chemistry between two main le...	fresh
...	...	...	...
383996	rotten	if there ever film that showcases hypocrisy of...	fresh
383997	rotten	listless troublingly familiar thriller last th...	rotten
383998	rotten	so i didnt think movie any good no shock there...	rotten
383999	rotten	those hoping for unintended laugh riot disappo...	rotten
384000	rotten	film that abuses its ridiculous premise never ...	fresh

96001 rows × 3 columns

```
dev_accuracy= sum(dev_data["smoothing_prediction"] == dev_data["Freshness"]) / len(dev_data)
print("accuarcy" ,dev_accuracy)
```

```
accuarcy 0.7959500421870606
```

```
a=classifying(dev_data,False)
```

```
dev_data["non_smoothing_prediction"] = a
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3672\2643654517.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-v
rsus-a-copy
  dev_data["non_smoothing_prediction"] = a
```

```
dev_data
```

	Freshness	Review	smoothing_prediction	non_smoothing_prediction
288000	rotten	tepid attempt at making alien abduction yarn i...	rotten	rotten
288001	fresh	perfectly serviceable raceagainsttheclock thri...	fresh	fresh
288002	fresh	poetic pulp movie made with joy	fresh	fresh
288003	fresh	thirst just best horror movie of year so far p...	fresh	fresh
288004	fresh	expect same easy chemistry between two main le...	fresh	fresh
...	...	...	...	...
383996	rotten	if there ever film that showcases hypocrisy of...	fresh	fresh
383997	rotten	listless troublingly familiar thriller last th...	rotten	rotten
383998	rotten	so i didnt think movie any good no shock there...	rotten	rotten
383999	rotten	those hoping for unintended laugh riot disappo...	rotten	rotten
384000	rotten	film that abuses its ridiculous premise never ...	fresh	fresh

96001 rows × 4 columns

```
dev_accuracy= sum(dev_data["non_smoothing_prediction"] == dev_data["Freshness"]) / len(dev_data)
print("accuarcy" ,dev_accuracy)
```

```
accuarcy 0.7947729711148842
```

Displaying the Top 10 words that predicts each class

```
positive_probs=prob_word_count_positive_smoothing/prob_word_count_negative_smoothing
negative_probs=prob_word_count_negative_smoothing/prob_word_count_positive_smoothing
fresh_top10=[]
rotten_top10=[]
for i in np.argsort(positive_probs):
    fresh_top10.append(words[i])
fresh_top10=fresh_top10[-10:]
for i in np.argsort(negative_probs):
    rotten_top10.append(words[i])
rotten_top10=rotten_top10[-10:]
```

```
for i in fresh_top10:
    if i in index_to_word:
        index=index_to_word[i]
        a=prob_word_count_positive_smoothing[index]
        b=prob_word_count_negative_smoothing[index]
        print(i)
        print("probability[class|word]:{} ".format(a/(a+b)))
```

```
reinvents
probability[class|word]:0.9654539952269497
cannily
probability[class|word]:0.9654539952269497
captivates
probability[class|word]:0.9666054558024604
koreedas
probability[class|word]:0.9666054558024604
tonic
probability[class|word]:0.9666054558024604
unadorned
probability[class|word]:0.9676826331511923
ida
probability[class|word]:0.9713758460411432
nimbly
probability[class|word]:0.9729230181668167
unmissable
probability[class|word]:0.9736355346988487
spiderverse
probability[class|word]:0.9755645525334212
```

```
for i in rotten_top10:
    if i in index_to_word:
        index=index_to_word[i]
        a=prob_word_count_positive_smoothing[index]
        b=prob_word_count_negative_smoothing[index]
        print(i)
        print("probability[class|word]:{} ".format(b/(a+b)))
```

```
drearily
probability[class|word]:0.9706423676059199
unfunny
probability[class|word]:0.9712346288653219
thirdrate
probability[class|word]:0.9714812025477481
lifeless
probability[class|word]:0.9719268325232326
feeble
probability[class|word]:0.9741857038970713
unexciting
probability[class|word]:0.975046221225918
flavorless
probability[class|word]:0.9778189801477026
squanders
probability[class|word]:0.9784184917496185
mirthless
probability[class|word]:0.9824888238849712
charmless
probability[class|word]:0.9930201603538501
```

Calculating the accuracy of the test data based on the optimal parameters we obtained while performing the development dataset. The accuracy on the test data set is 79.59%

```
a=classifying(test_data)
```

```
test_data["prediction"]=a
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3672\2370575236.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  test_data["prediction"]=a
```

```
test_accuracy= sum(test_data["prediction"] == test_data["Freshness"]) / len(test_data)
print("accuarcy" ,dev_accuracy)
```

```
accuarcy 0.7959500421870606
```

Challenges:

The main challenge, I was faced at Encoding. So, Where we need to remove the unwanted data from the text to get the better data to perform the our model. It took me some time to get better understand of the data to performing model encoding.

References: