

LAB ASSESSMENT: - 5

Name of Department: Department of Data Science

Course Code: MDA472

Course Name: NLP

Course Teacher: Prof. Niju P Joseph

Register Number: 2282409

Name: Krutika Gajanan Rajpure

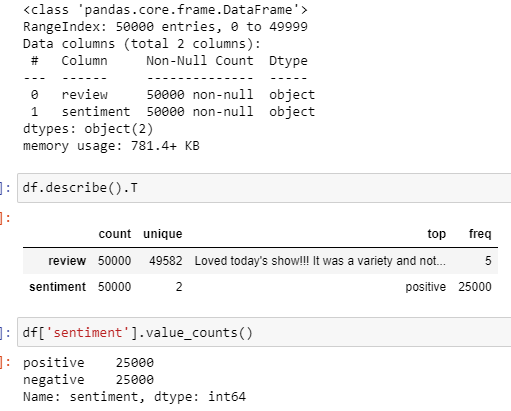
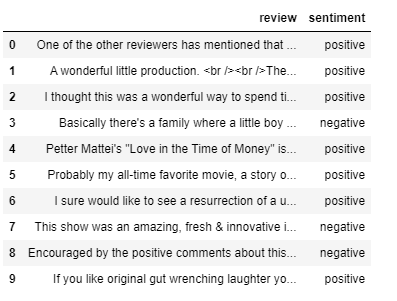
Date of submission: 15/08/2023

# Lab Assessment 5: Sentimental Analysis Using Naive-Bayes Classifier

A sentiment evaluation system for text analysis combines natural language processing (NLP) and laptop mastering methods to assign weighted sentiment scores to the entities, topics, issues, and categories inside a sentence or phrase. It tries to find and justify the sentiment of the person with respect to a given source of content.

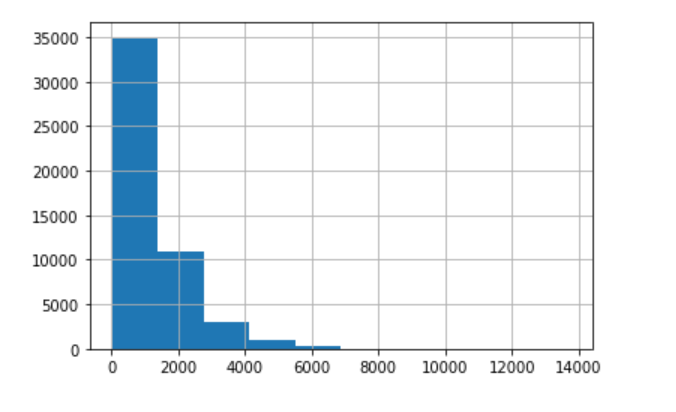
**Understanding the Data**

IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing.



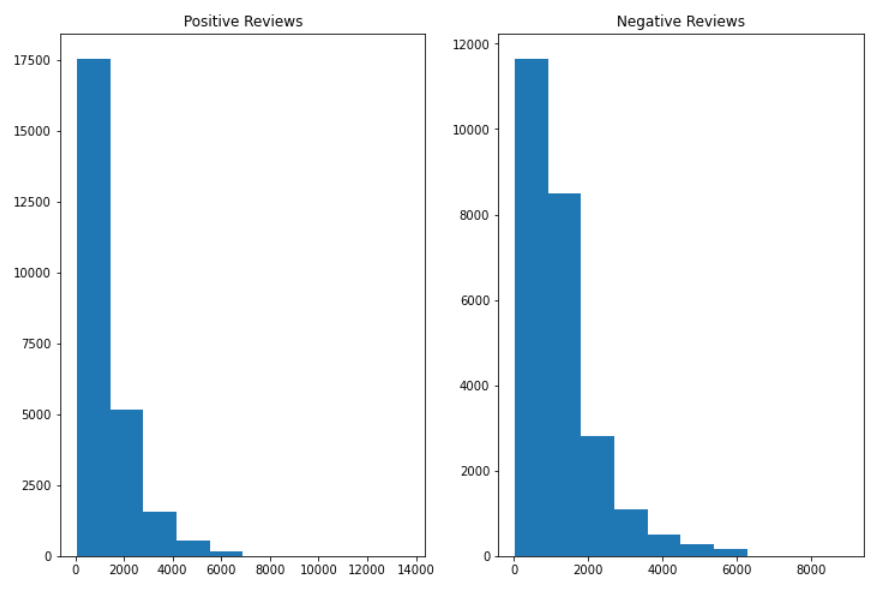
## **EDA and DATA PREPROCESSING**

 The number of characters present in each sentence. This can give us a rough idea about the movie's review.

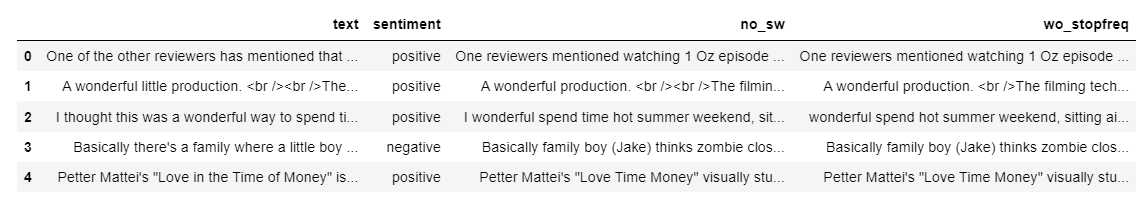
The histogram shows that reviews range from 10 to 14000 characters and generally, it is between 10 to 1500 characters.

We can observe some insights from the graph above:

* In general, people comment fewer words in the positive review to compare with a negative review
* However, the range of word for positive review is bigger than the range of negative review. It means in some cases, people give long comments for excellent movies and people could less criticise for bad movies



## **PREPROCESSING**



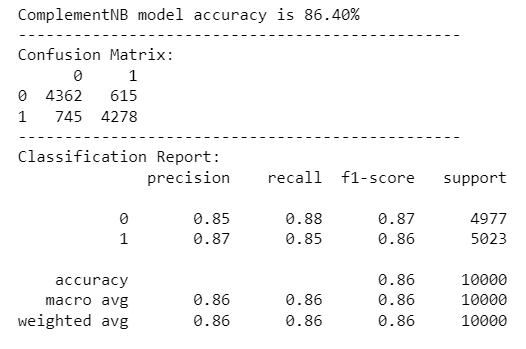
We will now split the data for training and testing to check how well our model has performed.

Also we will randomize the data in case our data includes all positive first and then all negative or some other kind of bias.

We will use: scikit\_learn's train\_test\_split() for splitting the text\_count (which contains our X) and dataset['Sentiment'] (this contains Y).

# Navies Bayes Modelling:

Different Navies Bayes Model: ComplementNB, MultinominalNB, BernoulliNB and see how accuracy each model can be:



Interpret the result:

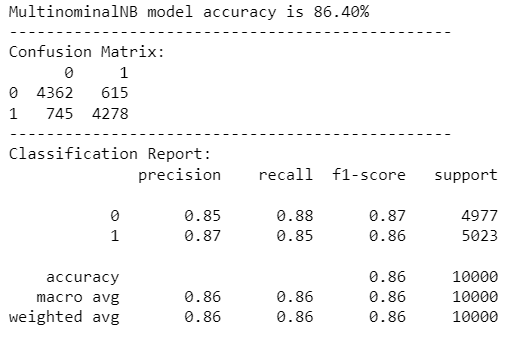
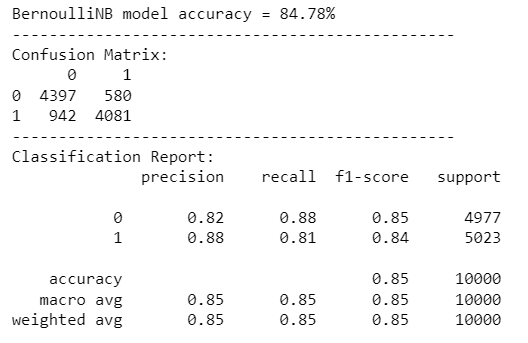
**Accuracy score:** is 86.33 % for the complementNB model. for each of 100 total number of prediction, in average, our model can predict 86 cases correctly

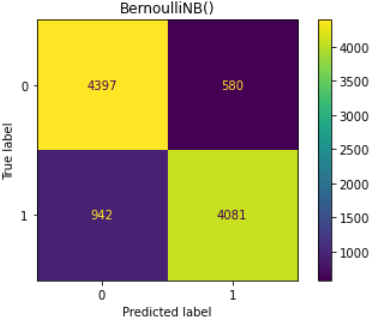
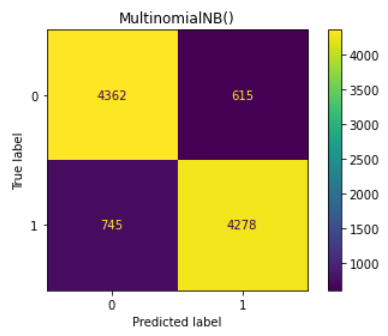
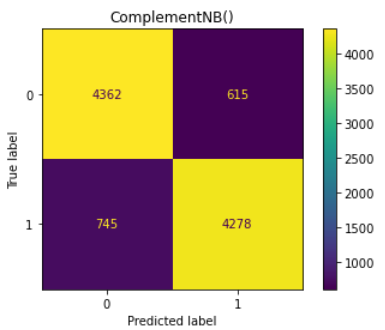
* Classification Report:

**Precision:**

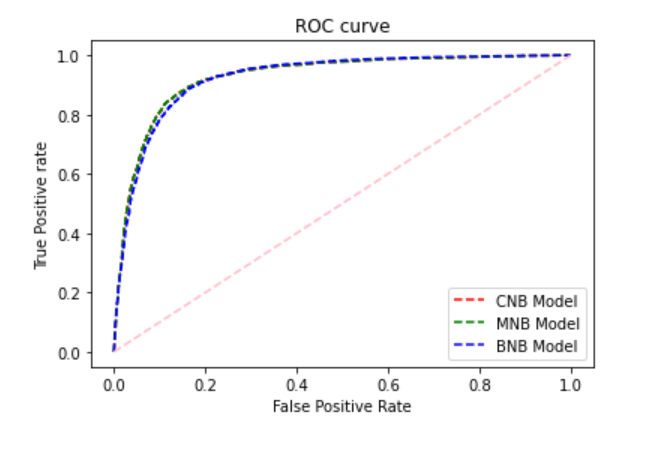
* Negative review: the ratio correctly predicted negative review observations to the total predicted negative review observations is 0.85
* Positive review: the ratio correctly predicted positive review observations to the total predicted positive review observations is 0.87

**Recall:**

* Negative review: the ratio of correctly predicted negative review observations to the all observations in actual class is 0.87
* Positive review: the ratio of correctly predicted positive review observations to the all observations in actual class is 0.85
* F1 Score is the weighted average of Precision and Recall. In both case positive and negative reviews, F1 scores are equal 0.86



The ROC curve visually depicts the trade-off between sensitivity and specificity for different classification thresholds. A model with a higher AUC score generally has better discriminatory power. The plot shows how well each model can distinguish between the positive and negative classes, and the random curve serves as a baseline comparison.



# Code:

#https://www.kaggle.com/code/ankumagawa/sentimental-analysis-using-naive-bayes-classifier

#Importing

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** plt

**import** plotly**.**express **as** px

**from** wordcloud **import** WordCloud

**import** nltk

**import** re

**import** string

**from** nltk**.**corpus **import** stopwords

nltk**.**download**(**'punkt'**)**

nltk**.**download**(**'stopwords'**)**

**from** nltk**.**tokenize **import** word\_tokenize

**from** nltk**.**stem **import** WordNetLemmatizer

stop\_words **=** stopwords**.**words**()**

pip install wordcloud

**from** wordcloud **import** WordCloud

df**=**pd**.**read\_csv**(**'IMDB Dataset.csv'**)**

df**.**head**(**10**)**

df**.**info**()**

df**.**describe**().**T

df**[**'sentiment'**].**value\_counts**()**

df**[**'review'**].str.len().**hist**()**

fig**,(**ax1**,**ax2**)=**plt**.**subplots**(**1**,**2**,**figsize**=(**12**,**8**))**

ax1**.**hist**(**df**[**df**[**'sentiment'**]==**'positive'**][**'review'**].str.len())**

ax1**.**set\_title**(** 'Positive Reviews'**)**

ax2**.**hist**(**df**[**df**[**'sentiment'**]==**'negative'**][**'review'**].str.len())**

ax2**.**set\_title**(** 'Negative Reviews'**)**

The wordcloud graphs **in** both negative **and** postitive comments don't show meaningful result.

That's is the reason why text preprosessing is needed.

# Prepocessing

df**.**rename**(**columns**={**'review'**:**'text'**},** inplace **=** **True)**

df

**def** cleaning**(**text**):**

# converting to lowercase, removing URL links, special characters, punctuations...

text **=** text**.**lower**()** # converting to lowercase

text **=** re**.**sub**(**'https?://\S+|www\.\S+'**,** ''**,** text**)** # removing URL links

text **=** re**.**sub**(**r"\b\d+\b"**,** ""**,** text**)** # removing number

text **=** re**.**sub**(**'<.\*?>+'**,** ''**,** text**)** # removing special characters,

text **=** re**.**sub**(**'[%s]' **%** re**.**escape**(**string**.**punctuation**),** ''**,** text**)** # punctuations

text **=** re**.**sub**(**'\n'**,** ''**,** text**)**

text **=** re**.**sub**(**'[’“”…]'**,** ''**,** text**)**

#removing emoji:

emoji\_pattern **=** re**.compile(**"["

u"\U0001F600-\U0001F64F" # emoticons

u"\U0001F300-\U0001F5FF" # symbols & pictographs

u"\U0001F680-\U0001F6FF" # transport & map symbols

u"\U0001F1E0-\U0001F1FF" # flags (iOS)

u"\U00002702-\U000027B0"

u"\U000024C2-\U0001F251"

"]+"**,** flags**=**re**.**UNICODE**)**

text **=** emoji\_pattern**.**sub**(**r''**,** text**)**

# removing short form:

text**=**re**.**sub**(**"isn't"**,**'is not'**,**text**)**

text**=**re**.**sub**(**"he's"**,**'he is'**,**text**)**

text**=**re**.**sub**(**"wasn't"**,**'was not'**,**text**)**

text**=**re**.**sub**(**"there's"**,**'there is'**,**text**)**

text**=**re**.**sub**(**"couldn't"**,**'could not'**,**text**)**

text**=**re**.**sub**(**"won't"**,**'will not'**,**text**)**

text**=**re**.**sub**(**"they're"**,**'they are'**,**text**)**

text**=**re**.**sub**(**"she's"**,**'she is'**,**text**)**

text**=**re**.**sub**(**"There's"**,**'there is'**,**text**)**

text**=**re**.**sub**(**"wouldn't"**,**'would not'**,**text**)**

text**=**re**.**sub**(**"haven't"**,**'have not'**,**text**)**

text**=**re**.**sub**(**"That's"**,**'That is'**,**text**)**

text**=**re**.**sub**(**"you've"**,**'you have'**,**text**)**

text**=**re**.**sub**(**"He's"**,**'He is'**,**text**)**

text**=**re**.**sub**(**"what's"**,**'what is'**,**text**)**

text**=**re**.**sub**(**"weren't"**,**'were not'**,**text**)**

text**=**re**.**sub**(**"we're"**,**'we are'**,**text**)**

text**=**re**.**sub**(**"hasn't"**,**'has not'**,**text**)**

text**=**re**.**sub**(**"you'd"**,**'you would'**,**text**)**

text**=**re**.**sub**(**"shouldn't"**,**'should not'**,**text**)**

text**=**re**.**sub**(**"let's"**,**'let us'**,**text**)**

text**=**re**.**sub**(**"they've"**,**'they have'**,**text**)**

text**=**re**.**sub**(**"You'll"**,**'You will'**,**text**)**

text**=**re**.**sub**(**"i'm"**,**'i am'**,**text**)**

text**=**re**.**sub**(**"we've"**,**'we have'**,**text**)**

text**=**re**.**sub**(**"it's"**,**'it is'**,**text**)**

text**=**re**.**sub**(**"don't"**,**'do not'**,**text**)**

text**=**re**.**sub**(**"that´s"**,**'that is'**,**text**)**

text**=**re**.**sub**(**"I´m"**,**'I am'**,**text**)**

text**=**re**.**sub**(**"it’s"**,**'it is'**,**text**)**

text**=**re**.**sub**(**"she´s"**,**'she is'**,**text**)**

text**=**re**.**sub**(**"he’s'"**,**'he is'**,**text**)**

text**=**re**.**sub**(**'I’m'**,**'I am'**,**text**)**

text**=**re**.**sub**(**'I’d'**,**'I did'**,**text**)**

text**=**re**.**sub**(**"he’s'"**,**'he is'**,**text**)**

text**=**re**.**sub**(**'there’s'**,**'there is'**,**text**)**

**return** text

dt **=** df**[**'text'**].**apply**(**cleaning**)**

df**[**'sentiment'**]**

dt **=** pd**.**DataFrame**(**df**)**

dt**[**'sentiment'**]=**df**[**'sentiment'**]**

dt

# remove stop word:

dt**[**'no\_sw'**]** **=** dt**[**'text'**].**apply**(lambda** x**:** ' '**.**join**([**word **for** word **in** x**.**split**()** **if** word **not** **in** **(**stop\_words**)]))**

dt

#Working with the most Frequent Words:

**from** collections **import** Counter

cnt **=** Counter**()**

**for** text **in** dt**[**"no\_sw"**].**values**:**

**for** word **in** text**.**split**():**

cnt**[**word**]** **+=** 1

cnt**.**most\_common**(**10**)**

temp **=** pd**.**DataFrame**(**cnt**.**most\_common**(**10**))**

temp**.**columns**=[**'word'**,** 'count'**]**

temp

px**.**bar**(**temp**,** x**=**"count"**,** y**=**"word"**,** title**=**'Commmon Words in Text'**,** orientation**=**'h'**,**

width**=**700**,** height**=**700**)**

# Remove the most frequent words:

FREQWORDS **=** **set([**w **for** **(**w**,** wc**)** **in** cnt**.**most\_common**(**10**)])**

**def** remove\_freqwords**(**text**):**

"""custom function to remove the frequent words"""

**return** " "**.**join**([**word **for** word **in** **str(**text**).**split**()** **if** word **not** **in** FREQWORDS**])**

dt**[**"wo\_stopfreq"**]** **=** dt**[**"no\_sw"**].**apply**(lambda** text**:** remove\_freqwords**(**text**))**

dt**.**head**()**

dt**[**'no\_sw'**].**loc**[**5**]**

dt**[**'wo\_stopfreq'**].**loc**[**5**]**

# Lemmatization: Lemmatization is converting the word to its base form or lemma by removing affixes from the inflected words.

# It helps to create better features for machine learning and NLP models hence it is an important preprocessing step.

wordnet\_lem **=** WordNetLemmatizer**()**

dt**[**'wo\_stopfreq\_lem'**]** **=** dt**[**'wo\_stopfreq'**].**apply**(**wordnet\_lem**.**lemmatize**)**

dt

# Tokenization

# create the cleaned data for the train-test split:

nb**=**dt**.**drop**(**columns**=[**'text'**,**'no\_sw'**,** 'wo\_stopfreq'**])**

nb**.**columns**=[**'sentiment'**,**'review'**]**

nb**.**sentiment **=** **[**0 **if** each **==** "negative" **else** 1 **for** each **in** nb**.**sentiment**]**

nb

tokenized\_review**=**nb**[**'review'**].**apply**(lambda** x**:** x**.**split**())**

tokenized\_review**.**head**(**5**)**

**from** sklearn**.**feature\_extraction**.**text **import** CountVectorizer

**from** nltk**.**tokenize **import** RegexpTokenizer

token **=** RegexpTokenizer**(**r'[a-zA-Z0-9]+'**)**

cv **=** CountVectorizer**(**stop\_words**=**'english'**,**ngram\_range **=** **(**1**,**1**),**tokenizer **=** token**.**tokenize**)**

text\_counts **=** cv**.**fit\_transform**(**nb**[**'review'**])**

text\_counts

nb**[**'review'**][**0**]**

count **=** CountVectorizer**()**

word\_count**=**count**.**fit\_transform**(**nb**[**'review'**])**

**print(**word\_count**)**

# Train-test-split

**from** sklearn**.**model\_selection **import** train\_test\_split

X**=**text\_counts

y**=**nb**[**'sentiment'**]**

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**X**,** y**,** test\_size**=**0.20**,**random\_state**=**30**)**

We will now split the data **for** training **and** testing to check how well our model has performed**.**

Also we will randomize the data **in** case our data includes **all** positive first **and** then **all** negative **or** some other kind of bias**.**

We will use**:** scikit\_learn's train\_test\_split() for splitting the text\_count (which contains our X) and dataset['Sentiment'] (this contains Y).

# Navies Bayes Modelling

**from** sklearn**.**naive\_bayes **import** ComplementNB

**from** sklearn**.**metrics **import** classification\_report**,** confusion\_matrix

CNB **=** ComplementNB**()**

CNB**.**fit**(**X\_train**,** y\_train**)**

**from** sklearn **import** metrics

predicted **=** CNB**.**predict**(**X\_test**)**

accuracy\_score **=** metrics**.**accuracy\_score**(**predicted**,** y\_test**)**

**print(**'ComplementNB model accuracy is'**,str(**'{:04.2f}'**.format(**accuracy\_score**\***100**))+**'%'**)**

**print(**'------------------------------------------------'**)**

**print(**'Confusion Matrix:'**)**

**print(**pd**.**DataFrame**(**confusion\_matrix**(**y\_test**,** predicted**)))**

**print(**'------------------------------------------------'**)**

**print(**'Classification Report:'**)**

**print(**classification\_report**(**y\_test**,** predicted**))**

**from** sklearn**.**naive\_bayes **import** MultinomialNB

MNB **=** MultinomialNB**()**

MNB**.**fit**(**X\_train**,** y\_train**)**

predicted **=** MNB**.**predict**(**X\_test**)**

accuracy\_score **=** metrics**.**accuracy\_score**(**predicted**,** y\_test**)**

**print(**'MultinominalNB model accuracy is'**,str(**'{:04.2f}'**.format(**accuracy\_score**\***100**))+**'%'**)**

**print(**'------------------------------------------------'**)**

**print(**'Confusion Matrix:'**)**

**print(**pd**.**DataFrame**(**confusion\_matrix**(**y\_test**,** predicted**)))**

**print(**'------------------------------------------------'**)**

**print(**'Classification Report:'**)**

**print(**classification\_report**(**y\_test**,** predicted**))**

**from** sklearn**.**naive\_bayes **import** BernoulliNB

BNB **=** BernoulliNB**()**

BNB**.**fit**(**X\_train**,** y\_train**)**

predicted **=** BNB**.**predict**(**X\_test**)**

accuracy\_score\_bnb **=** metrics**.**accuracy\_score**(**predicted**,**y\_test**)**

**print(**'BernoulliNB model accuracy = ' **+** **str(**'{:4.2f}'**.format(**accuracy\_score\_bnb**\***100**))+**'%'**)**

**print(**'------------------------------------------------'**)**

**print(**'Confusion Matrix:'**)**

**print(**pd**.**DataFrame**(**confusion\_matrix**(**y\_test**,** predicted**)))**

**print(**'------------------------------------------------'**)**

**print(**'Classification Report:'**)**

**print(**classification\_report**(**y\_test**,** predicted**))**

**from** sklearn**.**metrics **import** plot\_confusion\_matrix

**import** warnings

warnings**.**filterwarnings**(**"ignore"**)**

k**=** **[**CNB**,** MNB**,** BNB**]**

**for** i **in** k**:**

plot\_confusion\_matrix**(**i**,** X\_test**,** y\_test**)**

plt**.**title**(**i**)**

plt**.**show**()**

**from** sklearn**.**metrics **import** roc\_curve

# predict probabilities for CNB, MNB, BNB models:

CNB\_prob **=** CNB**.**predict\_proba**(**X\_test**)**

MNB\_prob **=** MNB**.**predict\_proba**(**X\_test**)**

BNB\_prob **=** BNB**.**predict\_proba**(**X\_test**)**

# roc curve for models

fpr1**,** tpr1**,** thresh1 **=** roc\_curve**(**y\_test**,** CNB\_prob**[:,**1**],** pos\_label**=**1**)**

fpr2**,** tpr2**,** thresh2 **=** roc\_curve**(**y\_test**,** MNB\_prob**[:,**1**],** pos\_label**=**1**)**

fpr3**,** tpr3**,** thresh3 **=** roc\_curve**(**y\_test**,** BNB\_prob**[:,**1**],** pos\_label**=**1**)**

# roc curve for tpr = fpr

random\_probs **=** **[**0 **for** i **in** **range(len(**y\_test**))]**

p\_fpr**,** p\_tpr**,** \_ **=** roc\_curve**(**y\_test**,** random\_probs**,** pos\_label**=**1**)**

# auc scores

**from** sklearn**.**metrics **import** roc\_auc\_score

auc\_CNB **=** roc\_auc\_score**(**y\_test**,** CNB\_prob**[:,**1**])**

auc\_MNB **=** roc\_auc\_score**(**y\_test**,** MNB\_prob**[:,**1**])**

auc\_BNB **=** roc\_auc\_score**(**y\_test**,** BNB\_prob**[:,**1**])**

**print(**auc\_CNB**,** auc\_MNB**,**auc\_BNB**)**

# plot roc curves

plt**.**plot**(**fpr1**,** tpr1**,** linestyle**=**'--'**,**color**=**'red'**,** label**=**'CNB Model'**)**

plt**.**plot**(**fpr2**,** tpr2**,** linestyle**=**'--'**,**color**=**'green'**,** label**=**'MNB Model'**)**

plt**.**plot**(**fpr3**,** tpr3**,** linestyle**=**'--'**,**color**=**'blue'**,** label**=**'BNB Model'**)**

plt**.**plot**(**p\_fpr**,** p\_tpr**,** linestyle**=**'--'**,** color**=**'pink'**)**

# title

plt**.**title**(**'ROC curve'**)**

# x label

plt**.**xlabel**(**'False Positive Rate'**)**

# y label

plt**.**ylabel**(**'True Positive rate'**)**

plt**.**legend**(**loc**=**'best'**)**

plt**.**savefig**(**'ROC'**,**dpi**=**300**)**

plt**.**show**();**