**BUILDING A LANGUAGE DETECTION MODEL**

**USING NLP**

*Submitted by*

**SARIKA V S**

**122003226, B. Tech Computer Science and Engineering (3rd year)**

**Summary about the Base Paper:**

**Title:** Automatic Language Detection in texts

**Year:** April 2018

**Authors:** Tommi Jauhiainen, Marco Lui, Marcos Zampieri, Timothy Baldwin, Krister Linden

**Reference Link:** *https://www.researchgate.net/publication/324717246\_Automatic\_Language\_Identification\_in\_Texts\_A\_Survey*

**OBJECTIVE:**

The increase in the use of microblogging came along with the rapid growth on short linguistic data. On the other hand, machine learning is considered to be the new frontier to extract meaningful information out of large amount of raw data in an automated manner. This project was created in an attempt to identify the language in which a document, message, or sentence is written. Using the text, The model will be created which will be able to predict the given language. This is a solution for many artificial intelligence applications and computational linguists. These kinds of prediction systems are widely used in electronic devices such as mobiles, laptops, etc for machine translation, and also on robots. It helps in tracking and identifying multilingual documents too.

**DATASET:**

**No. of. records:** 10337

**Attributes:** Text, Language

The Language Detection dataset contains text details for 17 different languages.

Languages are:

* English
* Portuguese
* French
* Greek
* Dutch
* Spanish
* Japanese
* Russian
* Danish
* Italian
* Turkish
* Swedish
* Arabic
* Malayalam
* Hindi
* Tamil
* Kannada

**ABSTRACT**

Language identification (“LI”) is the problem of determining the natural language that

a document or part thereof is written in. Automatic LI has been extensively researched for

over fifty years. Today, LI is a key part of many text processing pipelines, as text processing

techniques generally assume that the language of the input text is known. Research in this

area has recently been especially active. This article provides a brief history of LI research,

and an extensive survey of the features and methods used in the LI literature. We describe

the features and methods using a unified notation, to make the relationships between

methods clearer. We discuss evaluation methods, applications of LI, as well as off-the-shelf

LI systems that do not require training by the end user. Finally, we identify open issues,

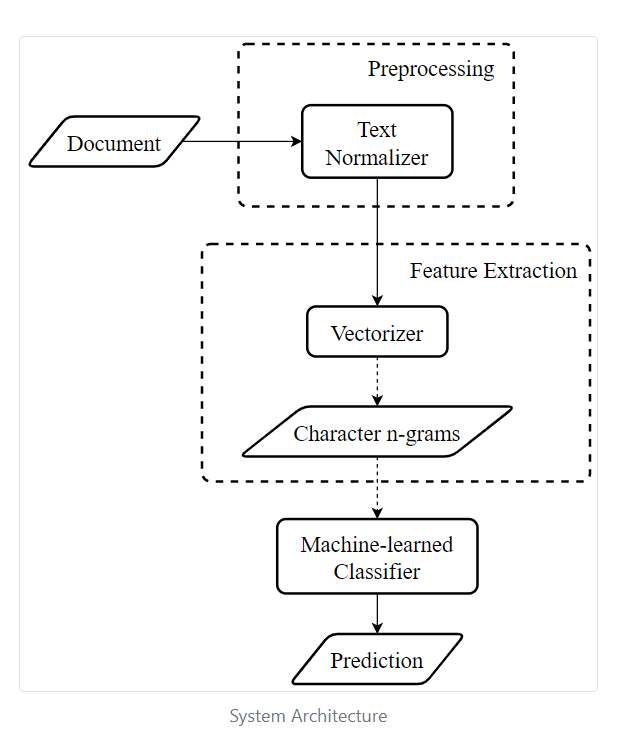
survey the work to date on each issue, and propose future directions for research in LI.

**KEY WORDS:** Machine Learning, Natural Language Processing, Tokenization

**PROPOSED WORK**

**Language Detection System:**

The system architecture can be viewed as a pipeline consisting of a text pre-processing module, a vectorizer that transforms text into character n-gram features, and a machine-learned classifier that predicts the language in which a document, message, or sentence is written. The following figure illustrates this architecture.



**SOURCE CODE**

**Importing libraries:**

import numpy as np

import pandas as pd

import re

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

import pickle

import tkinter as tk

warnings.simplefilter("ignore")

**Loading dataset:**

data = pd.read\_csv("Downloads/Language Detection.csv")

data.head(25)

data["Language"].value\_counts()

data.columns

data

**Seperating Depending and Independent Variables:**

#independent variable

X = data["Text"]

#dependent variable

y = data["Language"]

**Label Encoding:**

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y = le.fit\_transform(y)

**Text preprocessing:**

# creating a list for appending the preprocessed text

data\_list = []

# iterating through all the text

for text in X:

# removing the symbols and numbers

text = re.sub(r'[!@#$(),n"%^\*?:;~`0-9]', ' ', text)

text = re.sub(r'[[]]', ' ', text)

# converting the text to lower case

text = text.lower()

# appending to data\_list

data\_list.append(text)

**Bag of Words:**

**#** converting text into numerical form by creating a Bag of Words model using CountVectorizer.

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer()

X = cv.fit\_transform(data\_list).toarray()

X.shape

**Train & Test Splitting:**

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20)

**NAIVE BAYES MODEL:**

**MultiNomial NB:**

**Model Fitting & Model Evaluation:**

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(x\_train, y\_train)

y\_pred\_NB = model.predict(x\_test)

y\_pred\_NB

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report,precision\_score,recall\_score,f1\_score

ac\_NB = accuracy\_score(y\_test, y\_pred\_NB)

cm\_NB = confusion\_matrix(y\_test, y\_pred\_NB,)

pr\_NB = precision\_score(y\_test,y\_pred\_NB,average='macro')

rc\_NB = recall\_score(y\_test,y\_pred\_NB,average='macro')

f1\_NB = f1\_score(y\_test,y\_pred\_NB,average='macro')

print("Accuracy is :",ac\_NB)

print("Precision Score is :",pr\_NB)

print("Recall Score is :",rc\_NB)

print("F1 Score is",f1\_NB)

plt.figure(figsize=(15,10))

sns.heatmap(cm\_NB, annot = True)

plt.show()

**Gaussian NB:**

**Model Fitting & Model Evaluation:**

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(x\_train, y\_train)

y\_pred\_GNB = model.predict(x\_test)

y\_pred\_GNB

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report,precision\_score,recall\_score,f1\_score

ac\_GNB = accuracy\_score(y\_test, y\_pred\_GNB)

cm\_GNB = confusion\_matrix(y\_test, y\_pred\_GNB,)

pr\_GNB = precision\_score(y\_test,y\_pred\_GNB,average='macro')

rc\_GNB = recall\_score(y\_test,y\_pred\_GNB,average='macro')

f1\_GNB = f1\_score(y\_test,y\_pred\_GNB,average='macro')

print("Accuracy is :",ac\_GNB)

print("Precision Score is :",pr\_GNB)

print("Recall Score is :",rc\_GNB)

print("F1 Score is",f1\_GNB)

plt.figure(figsize=(15,10))

sns.heatmap(cm\_GNB, annot = True)

plt.show()

**Logistic Regression:**

**Model Fitting & Model Evaluation:**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(x\_train, y\_train)

y\_pred\_LR = model.predict(x\_test)

y\_pred\_LR

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report,precision\_score,recall\_score,f1\_score

ac\_LR = accuracy\_score(y\_test, y\_pred\_LR)

cm\_LR = confusion\_matrix(y\_test, y\_pred\_LR,)

pr\_LR = precision\_score(y\_test,y\_pred\_LR,average='macro')

rc\_LR = recall\_score(y\_test,y\_pred\_LR,average='macro')

f1\_LR = f1\_score(y\_test,y\_pred\_LR,average='macro')

print("Accuracy is :",ac\_LR)

print("Precision Score is :",pr\_LR)

print("Recall Score is :",rc\_LR)

print("F1 Score is",f1\_LR)

plt.figure(figsize=(15,10))

sns.heatmap(cm\_LR, annot = True)

plt.show()

**DecisionTree Classification:**

**Model Fitting & Model Evaluation:**

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred\_DTC = model.predict(x\_test)

y\_pred\_DTC

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report,precision\_score,recall\_score,f1\_score

ac\_DTC = accuracy\_score(y\_test, y\_pred\_DTC)

cm\_DTC = confusion\_matrix(y\_test, y\_pred\_DTC,)

pr\_DTC = precision\_score(y\_test,y\_pred\_DTC,average='macro')

rc\_DTC = recall\_score(y\_test,y\_pred\_DTC,average='macro')

f1\_DTC = f1\_score(y\_test,y\_pred\_DTC,average='macro')

print("Accuracy is :",ac\_DTC)

print("Precision Score is :",pr\_DTC)

print("Recall Score is :",rc\_DTC)

print("F1 Score is",f1\_DTC)

plt.figure(figsize=(15,10))

sns.heatmap(cm\_DTC, annot = True)

plt.show()

**Random Forest Classification:**

**Model Fitting & Model Evaluation:**

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_pred\_RFC = model.predict(x\_test)

y\_pred\_RFC

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report,precision\_score,recall\_score,f1\_score

ac\_RFC = accuracy\_score(y\_test, y\_pred\_RFC)

cm\_RFC = confusion\_matrix(y\_test, y\_pred\_RFC,)

pr\_RFC = precision\_score(y\_test,y\_pred\_RFC,average='macro')

rc\_RFC = recall\_score(y\_test,y\_pred\_RFC,average='macro')

f1\_RFC = f1\_score(y\_test,y\_pred\_RFC,average='macro')

print("Accuracy is :",ac\_RFC)

print("Precision Score is :",pr\_RFC)

print("Recall Score is :",rc\_RFC)

print("F1 Score is",f1\_RFC)

plt.figure(figsize=(15,10))

sns.heatmap(cm\_RFC, annot = True)

plt.show()

**COMPARING MODELS:**

**Comparing model accuracy:**

Accuracy = [ac\_NB,ac\_GNB,ac\_LR,ac\_DTC,ac\_RFC]

Methods = ['MNB','GNB','LR', 'DTC', 'RFC']

Accuracy\_pos = np.arange(len(Methods))

plt.bar(Accuracy\_pos, Accuracy)

plt.xticks(Accuracy\_pos, Methods)

plt.title('comparing the accuracy of each model')

plt.show()

**Comparing Model precision:**

Precision = [pr\_NB,pr\_GNB,pr\_LR,pr\_DTC,pr\_RFC]

Precision\_pos = np.arange(len(Methods))

plt.bar(Precision\_pos, Precision)

plt.xticks(Precision\_pos, Methods)

plt.title('comparing the precision of each model')

plt.show()

**Comparing Model recall:**

Recall = [rc\_NB,rc\_GNB,rc\_LR,rc\_DTC,rc\_RFC]

Recall\_pos = np.arange(len(Methods))

plt.bar(Recall\_pos, Recall)

plt.xticks(Recall\_pos, Methods)

plt.title('comparing the recall of each model')

plt.show()

**Comparing Model F1\_score:**

F1\_Score = [f1\_NB,f1\_GNB,f1\_LR,f1\_DTC,f1\_RFC]

F1\_Score\_pos = np.arange(len(Methods))

plt.bar(F1\_Score\_pos, F1\_Score)

plt.xticks(F1\_Score\_pos, Methods)

plt.title('comparing the F1 Score of each model')

plt.show()

**Deploying Model as a GUI:**

model = MultinomialNB()

model.fit(x\_train, y\_train)

def predict(text):

x = cv.transform([text]).toarray() # converting text to bag of words model (Vector)

lang = model.predict(x) # predicting the language

lang = le.inverse\_transform(lang) # finding the language corresponding the the predicted value

print("The language is in",lang[0]) # printing the language

la=str()

def onClick():

row= entertext.get()

print(row)

x = cv.transform([row]).toarray()

lang = model.predict(x)

lang = le.inverse\_transform(lang)

root2 = tk.Tk()

root2.title("Prediction Window")

la = lang[0]

print("The language is in",lang[0])

tk.Label(root2, text=la, font=("times new roman", 20), fg="white", bg="maroon", height=2).grid(row=0, column=1)

root = tk.Tk()

root.title("Language Detector:")

tk.Label(root,text="""Enter the text to detect which language it belongs to:""",font=("times new roman", 12)).grid(row=0)

tk.Label(root,text='Text:',padx=20, font=("times new roman", 12)).grid(row=1,column=0)

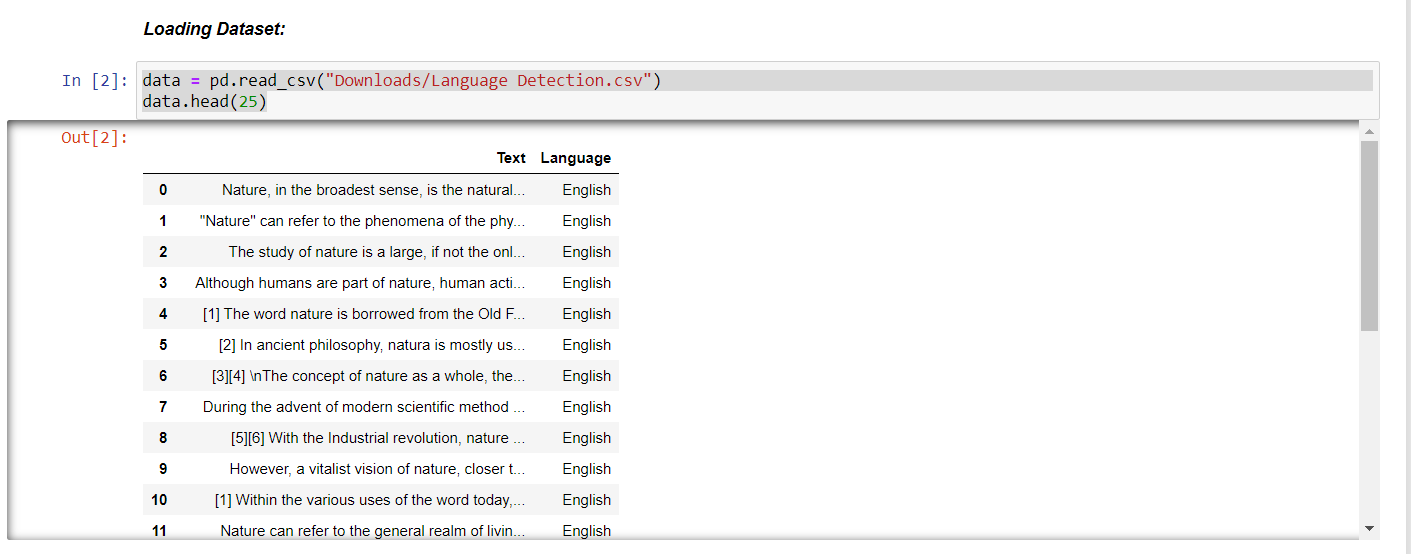
entertext = tk.StringVar()

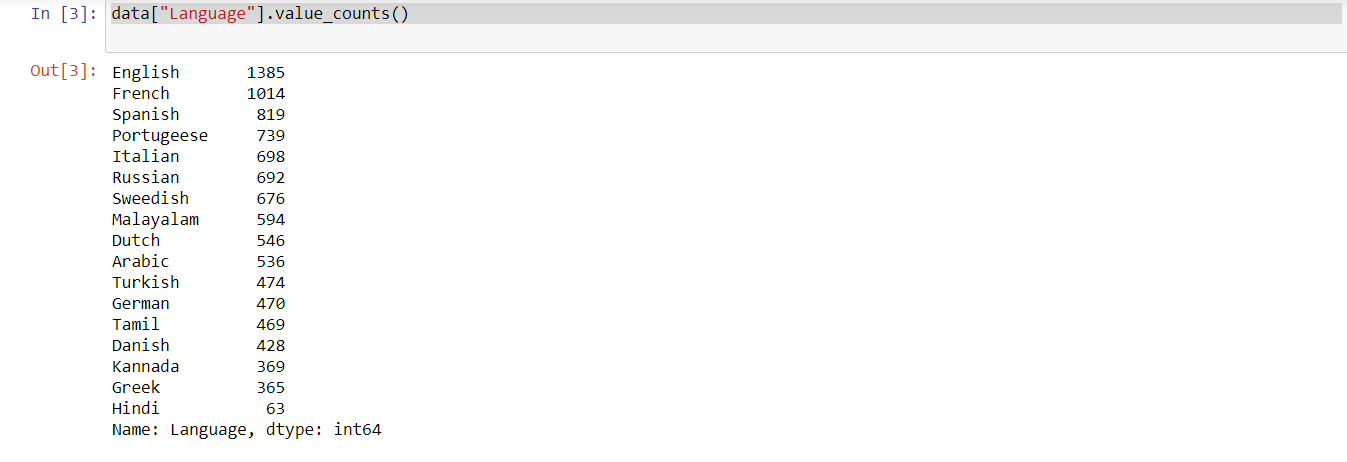
tk.Entry(root,textvariable=entertext).grid(row=1,column=1)

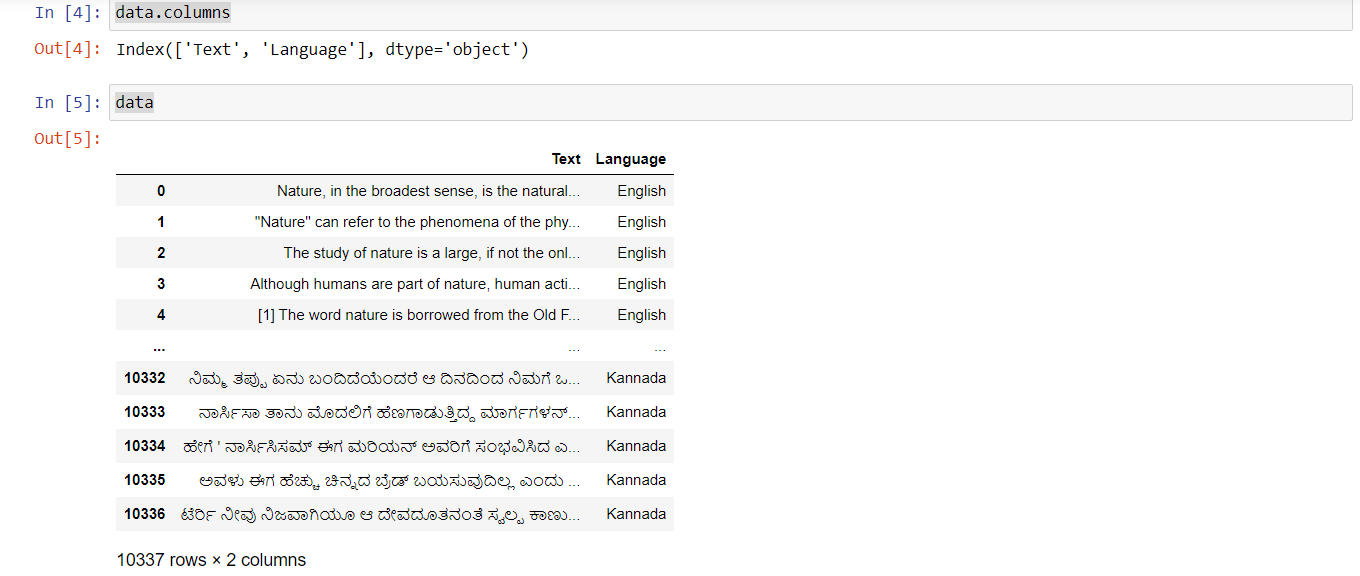
tk.Button(root, text='Predict', command=onClick).grid(row=11, column=1)

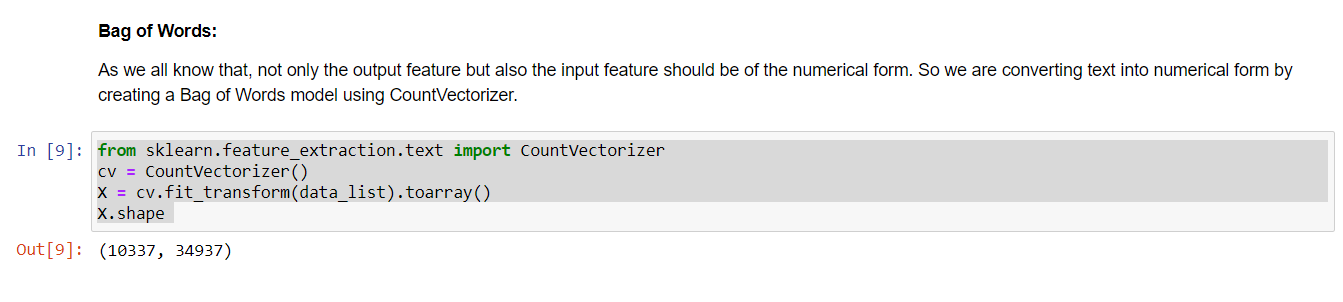
root.mainloop()

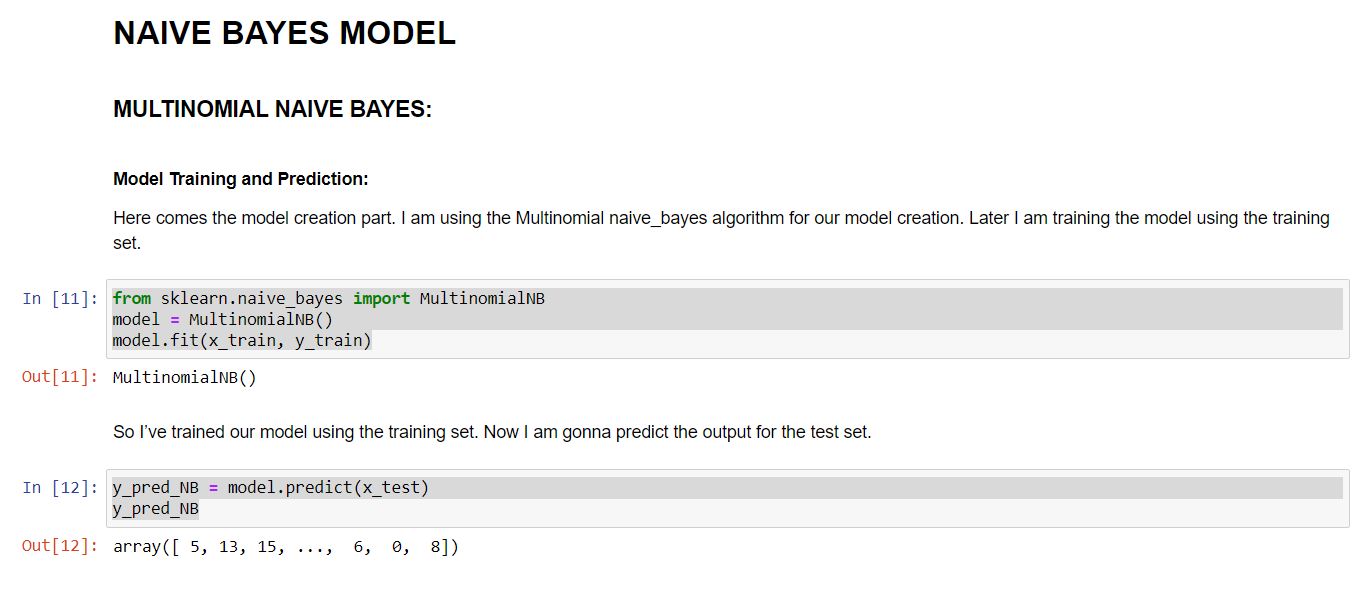
**OUTPUT SCREENSHOTS:**

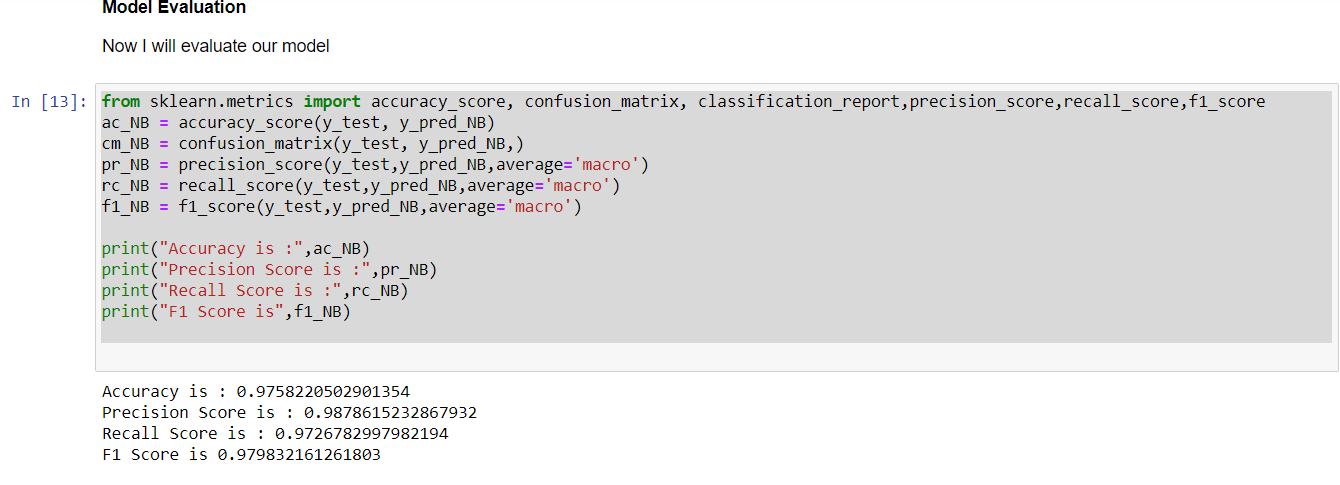
****

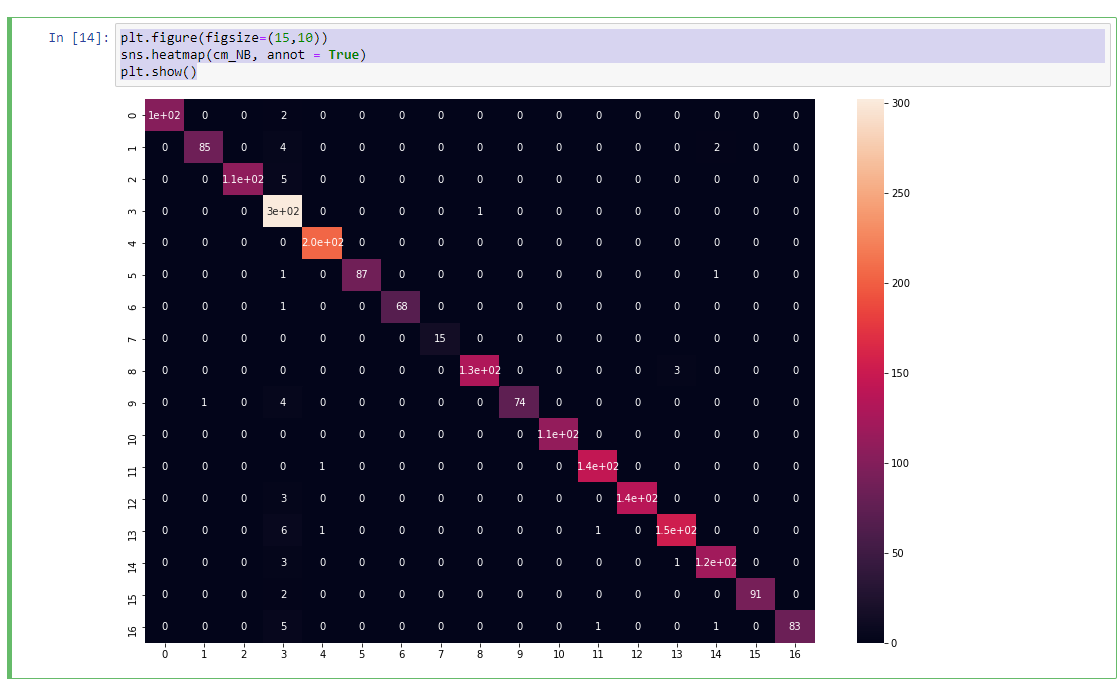


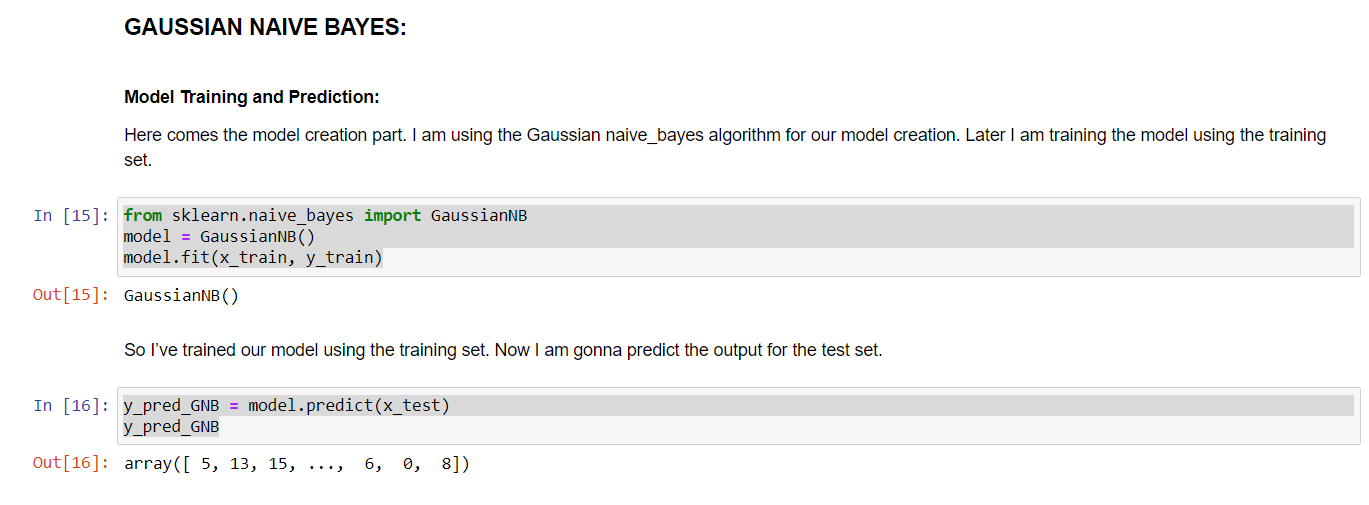




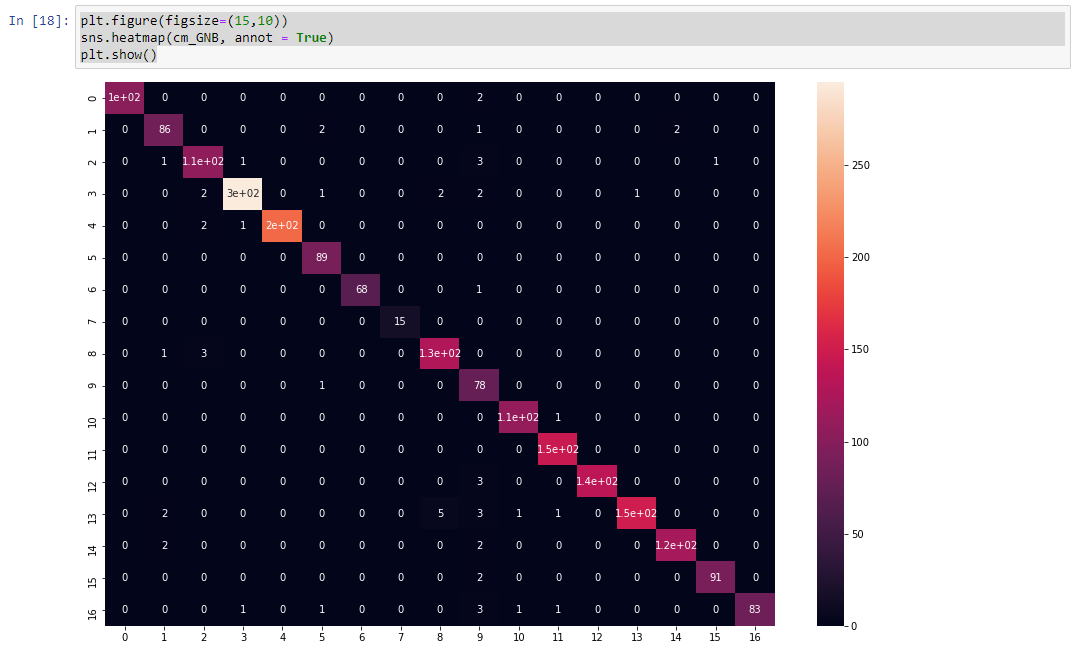


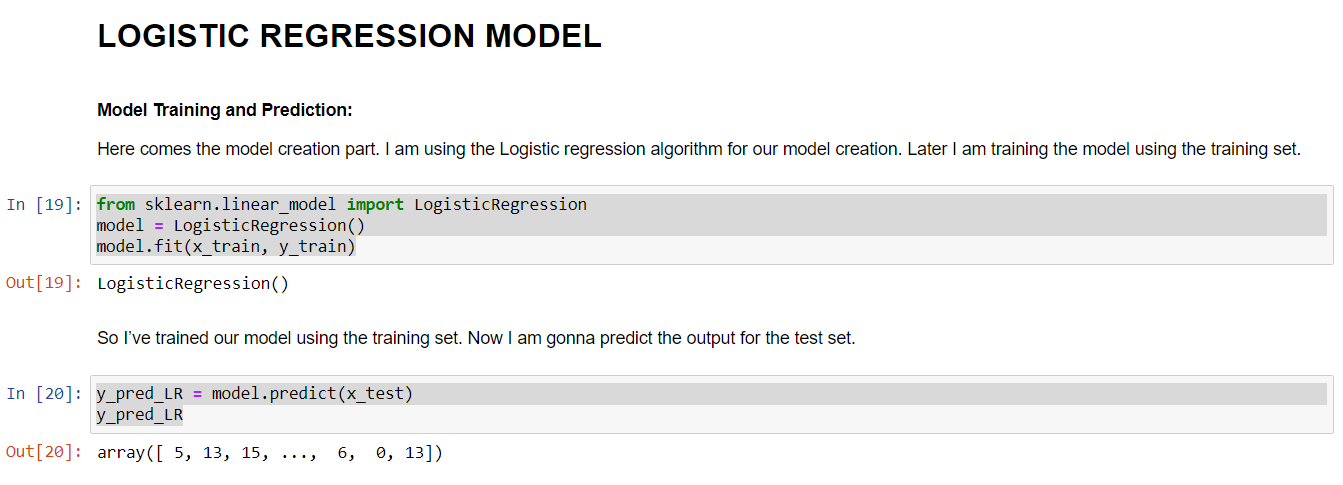


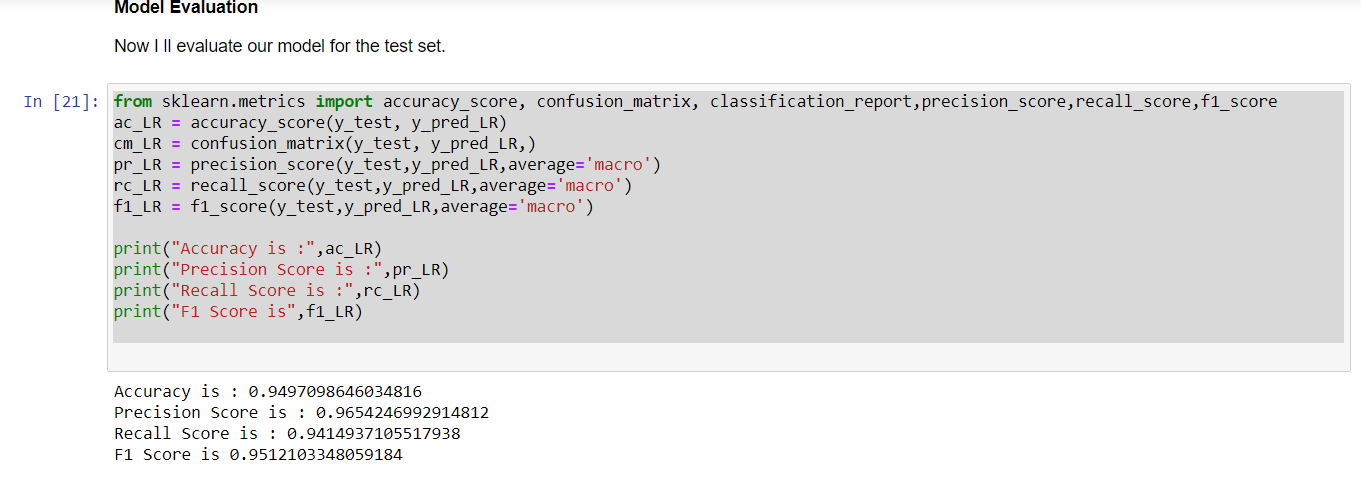


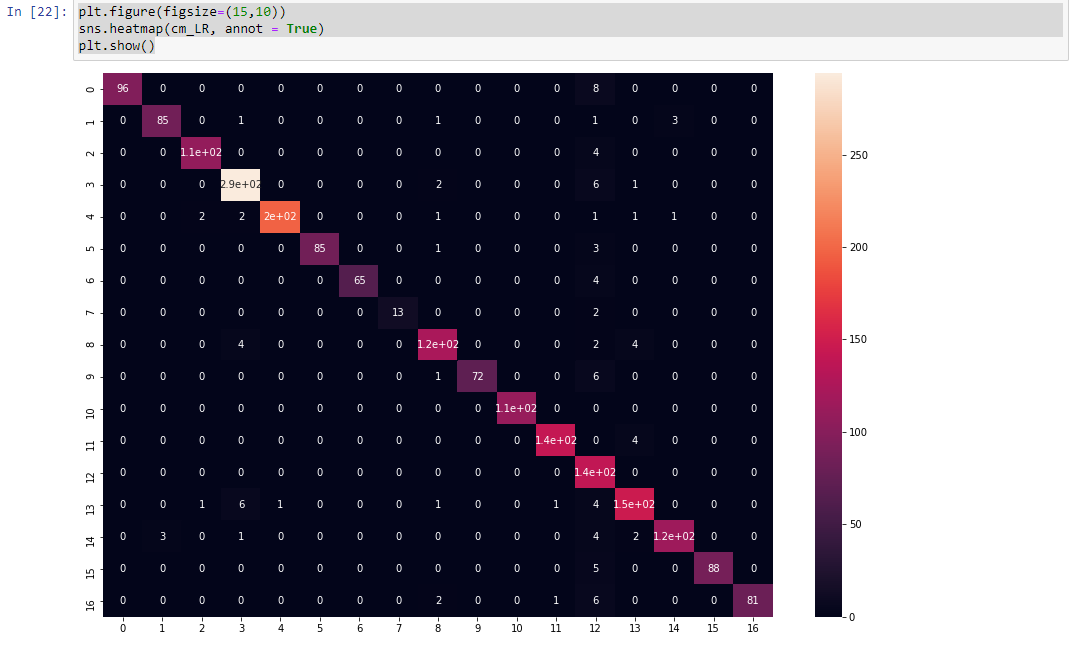


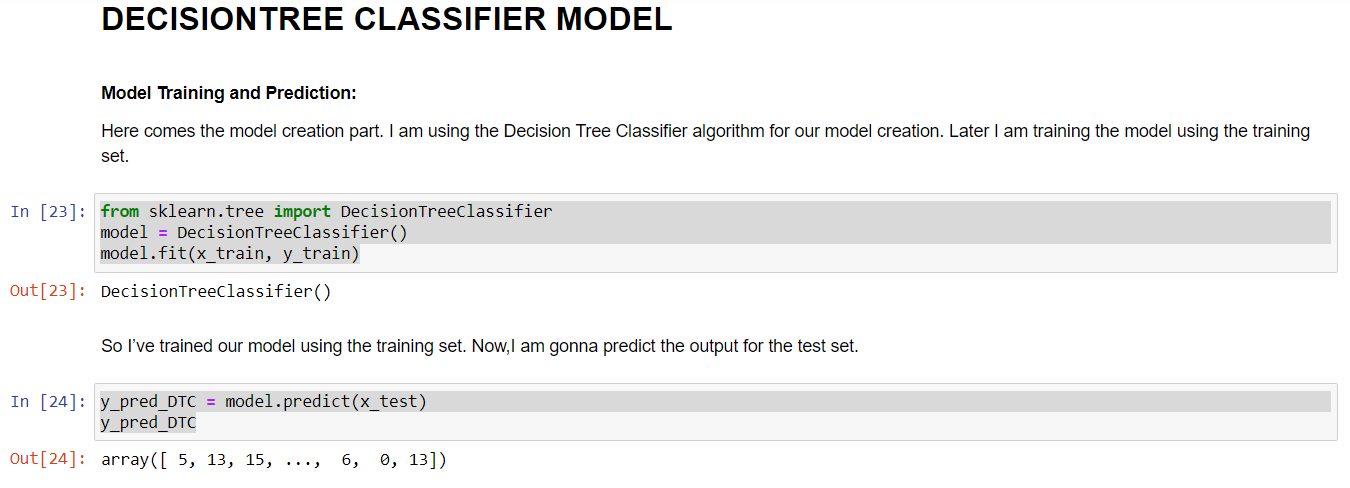


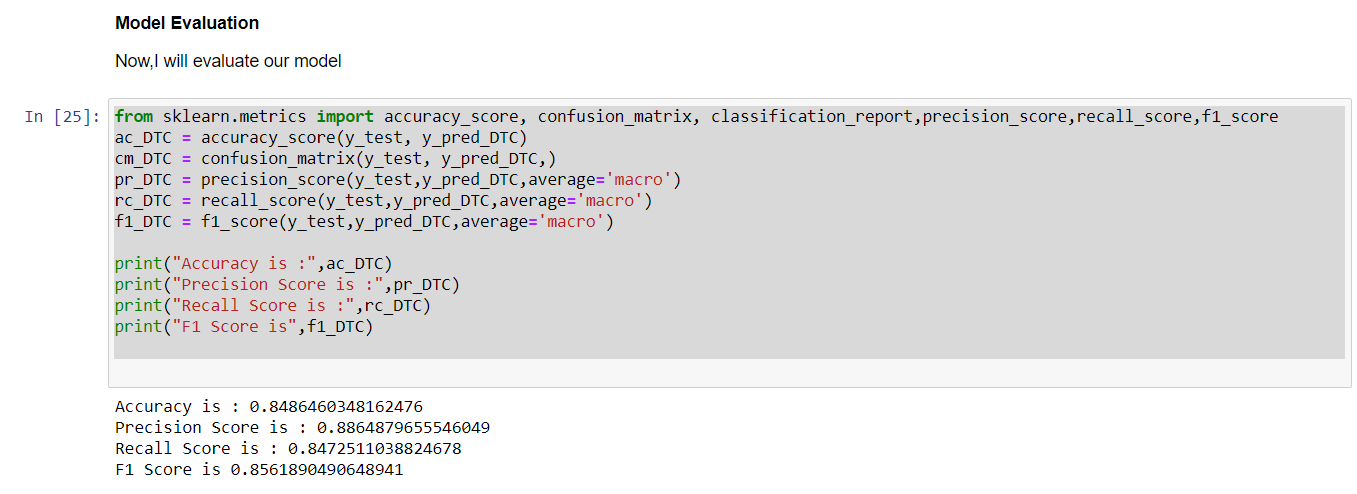


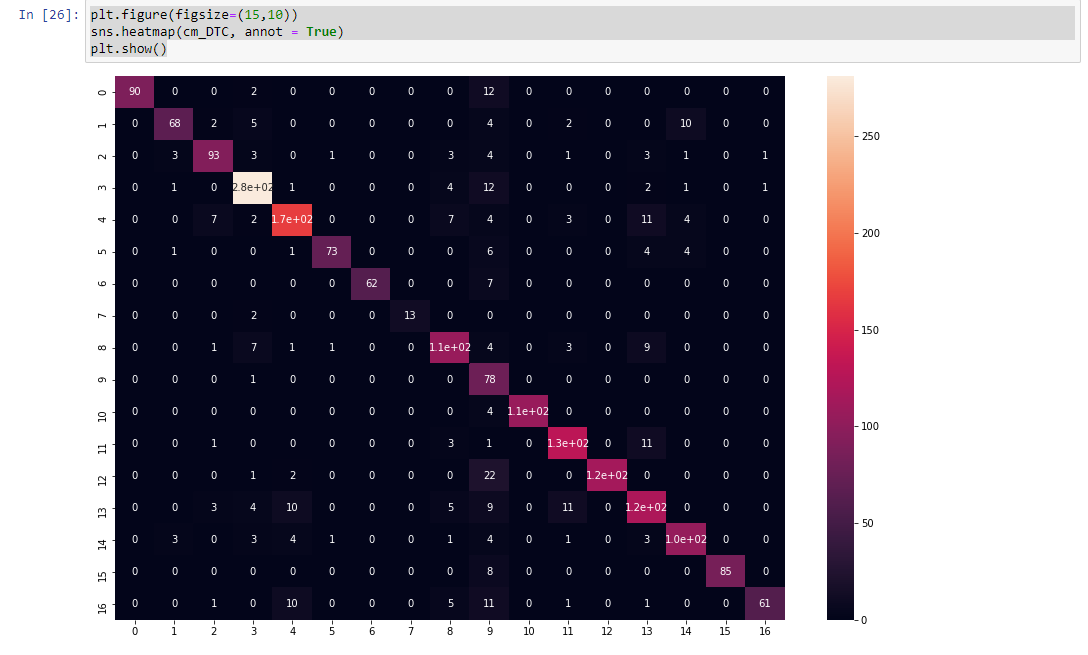


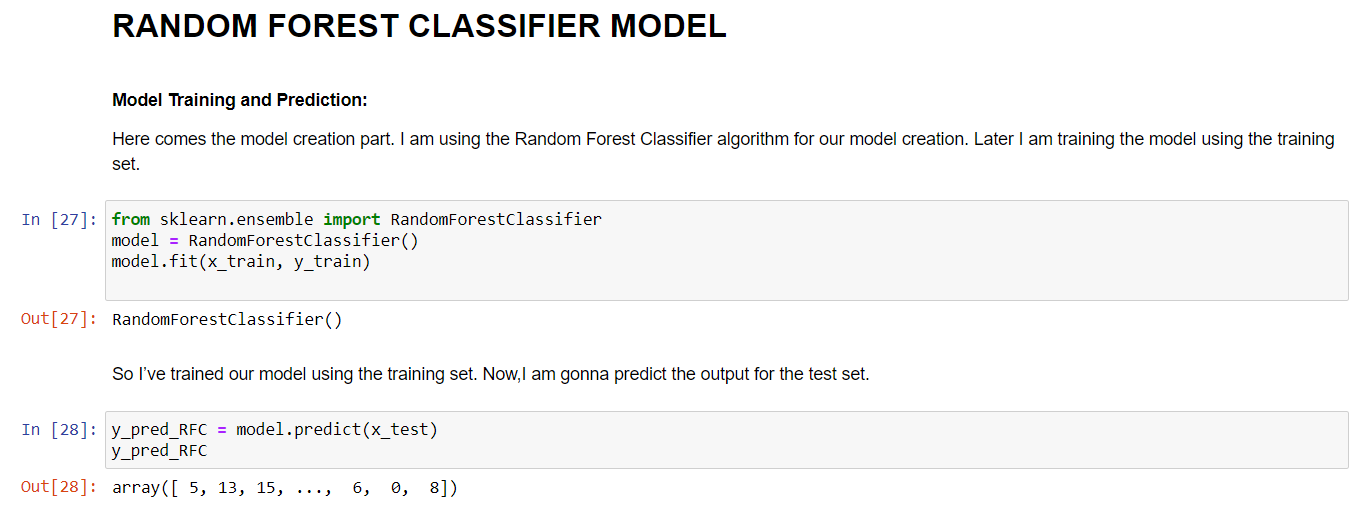


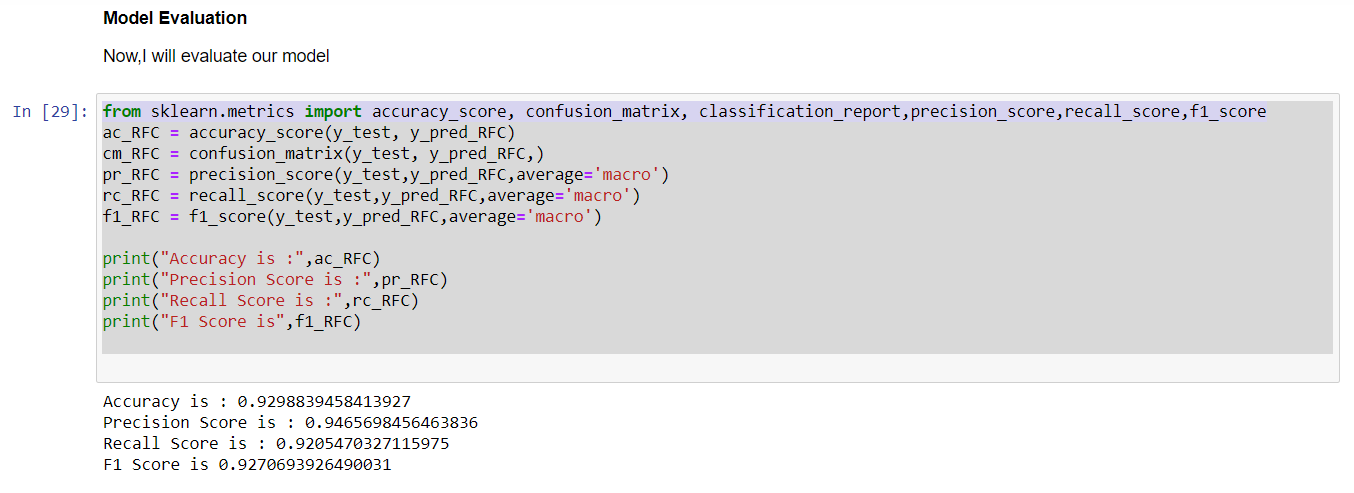


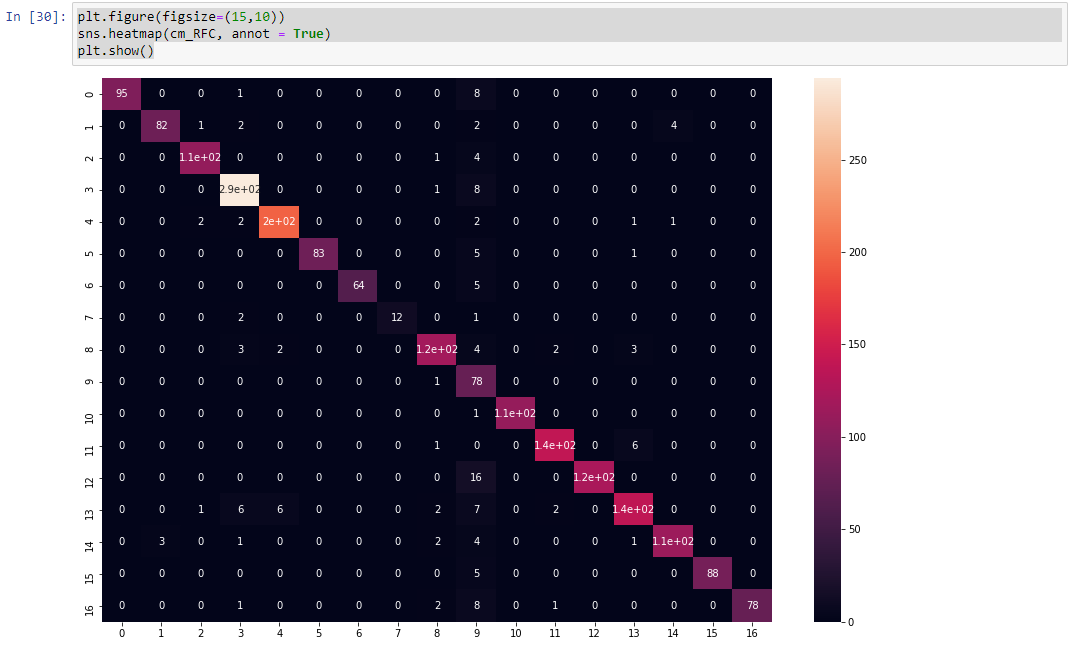


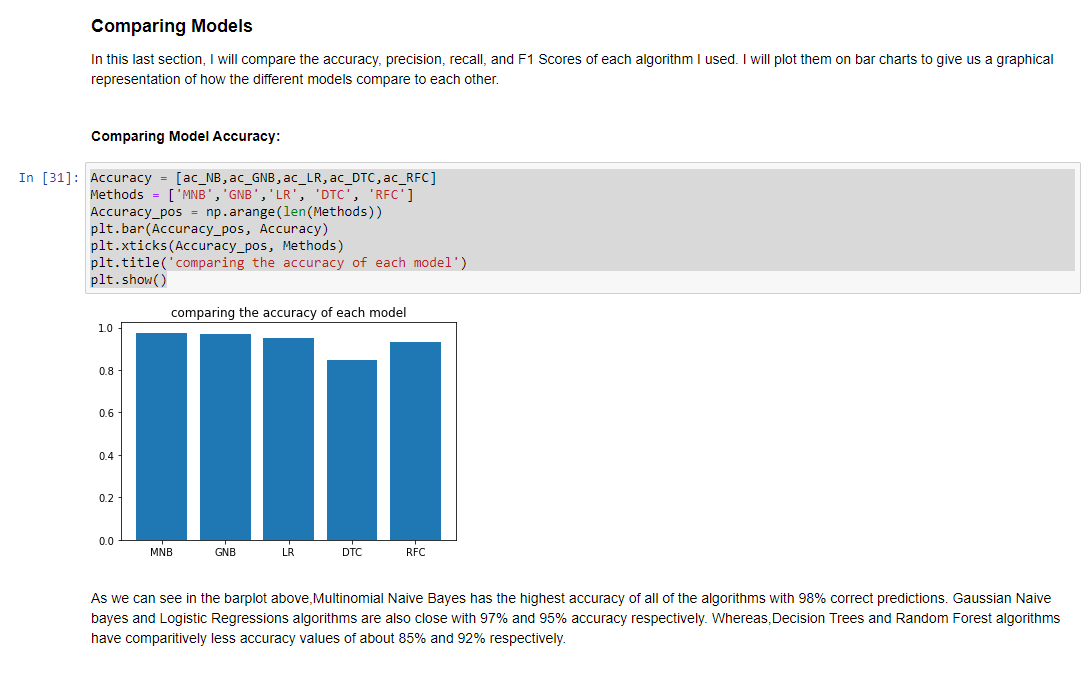


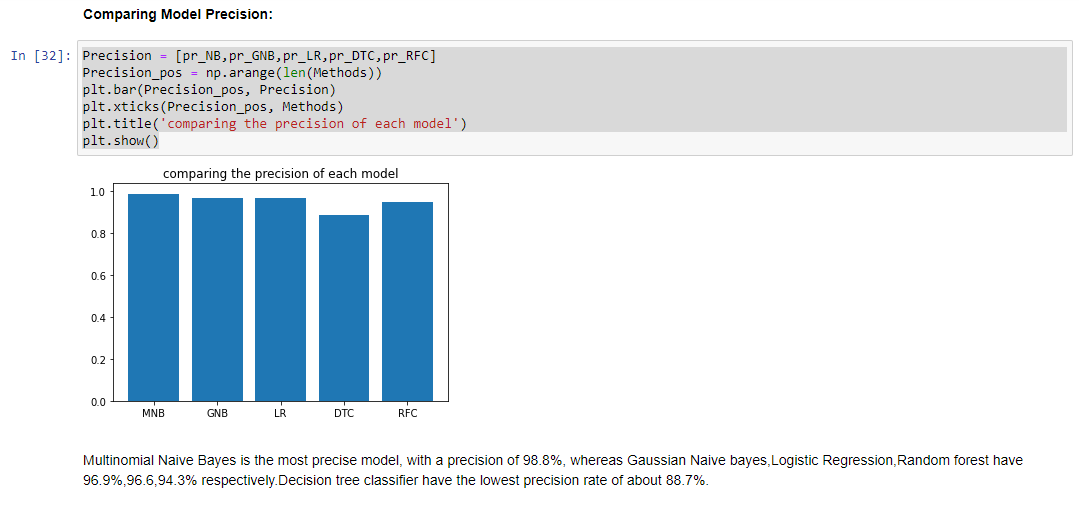
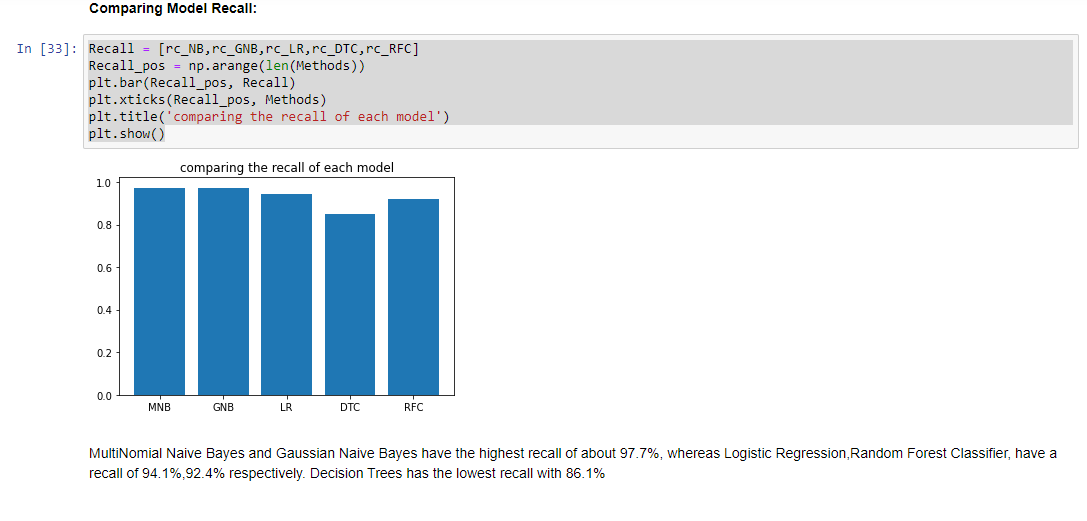


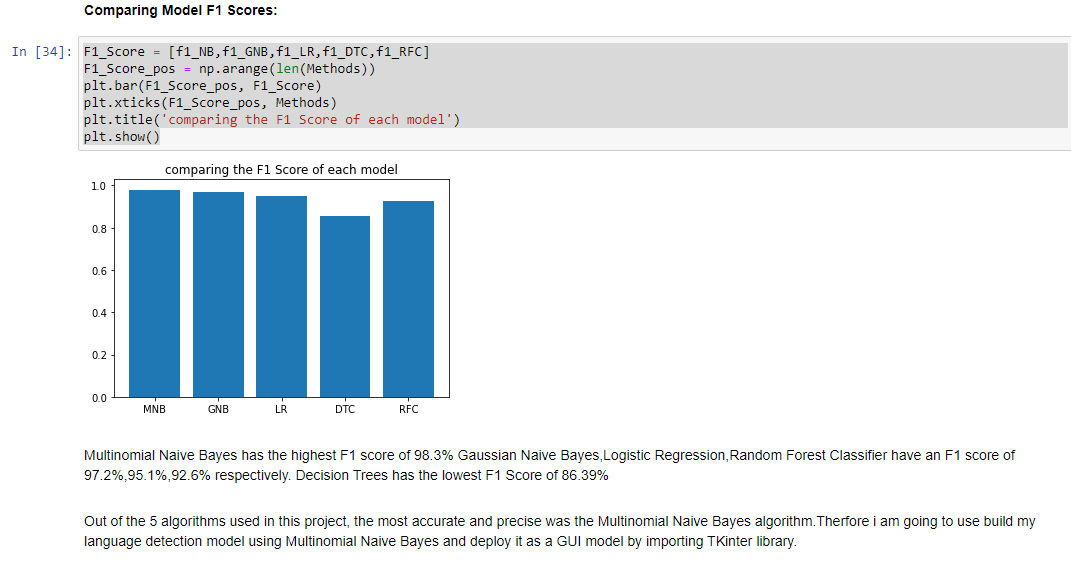




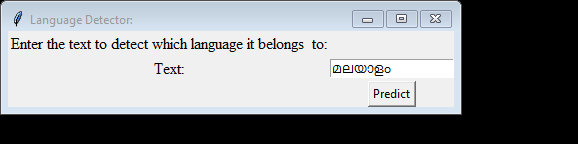




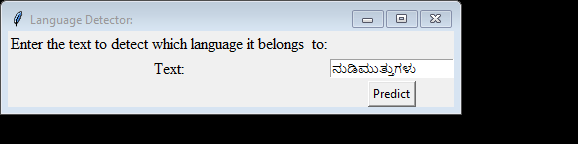


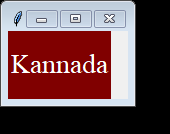


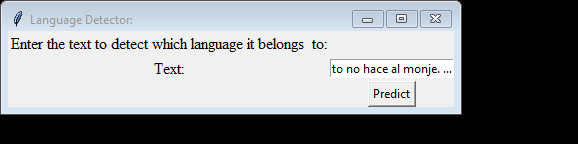


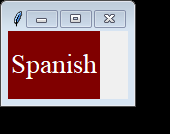


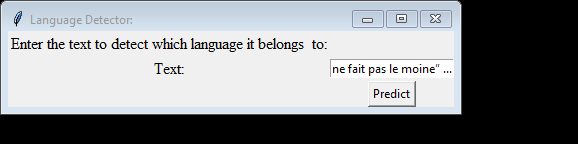


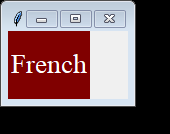


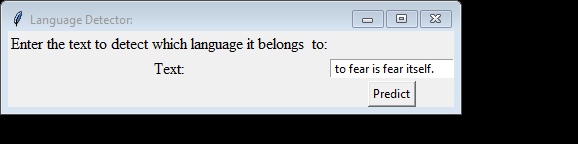


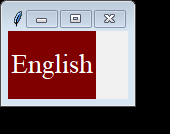












**CONCLUSION:**

Out of the 5 algorithms used in this project, the most accurate and precise was the Multinomial Naive Bayes algorithm. Therefore, I am going to use build my language detection model using Multinomial Naive Bayes and deploy it as a GUI model by importing TKinter library. In the end, the test accuracy of 98% leaves room for improvement. A more complicated approach could help us differentiate the languages that are more similar. We could also experiment with different models. Hopefully, this is a good starting point for your language identification experiments. Naive Bayes always proves to be a better model in such text classification problems, hence more accurate results we get. I have presented a Naïve Bayes based language identification scheme that achieves near perfect accuracy in classifying dissimilar languages and about 98% accuracy on highly similar languages. And expanding the corpus of these languages using external sources did not help much mainly because no n-grams of words that are unique to certain languages were ingested by the expanded part of the corpus. At this point, I think, further improvement can only be achieved by designing rule-based features.