# Naive Bayes

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Anandakrishnan k v Batch ID:** 19042021

**Topic: Naïve Bayes**

**Grading Guidelines:**

**1. An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered for evaluation.**

**2. Assignments submitted after the deadline will affect your grades.**

**Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ans** | **Date** |  |  | **Ans** | **Date** |
| Correct | On time | A | 100 |  |  |
| 80% & above | On time | B | 85 | Correct | Late |
| 50% & above | On time | C | 75 | 80% & above | Late |
| 50% & below | On time | D | 65 | 50% & above | Late |
|  |  | E | 55 | 50% & below |  |
| Copied/No Submission |  | F | 45 |  |  |

* **Grade A: (>= 90):** When all assignments are submitted on or before the given deadline.
* **Grade B: (>= 80 and < 90):** 
  + When assignments are submitted on time but less than 80% of problems are completed.

(OR)

* + All assignments are submitted after the deadline.
* **Grade C: (>= 70 and < 80):** 
  + When assignments are submitted on time but less than 50% of the problems are completed.

(OR)

* + Less than 80% of problems in the assignments are submitted after the deadline.
* **Grade D: (>= 60 and < 70):**
  + Assignments submitted after the deadline and with 50% or less problems.
* **Grade E: (>= 50 and < 60):** 
  + Less than 30% of problems in the assignments are submitted after the deadline.

(OR)

* + Less than 30% of problems in the assignments are submitted before the deadline.
* **Grade F: (< 50):** No submission (or) malpractice.

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**



**2.1 Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

1. **Data Pre-processing**

**3.1 Data Cleaning, Feature Engineering, etc.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the scaled data (try multiple options).**
   2. **Build a Naïve Bayes model.**

**5.3 Validate the model with test data and obtain a confusion matrix, get precision, recall, and accuracy from it.**

**5.4 Tune the model and improve the accuracy**

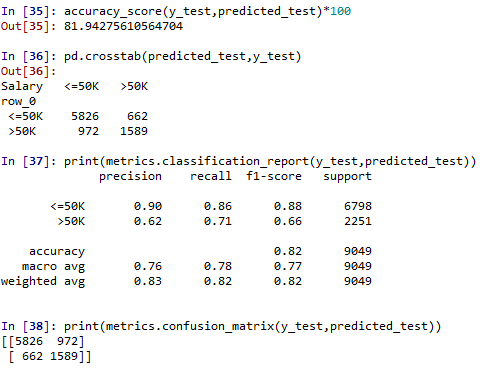
**6. Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

**Problem Statement:**

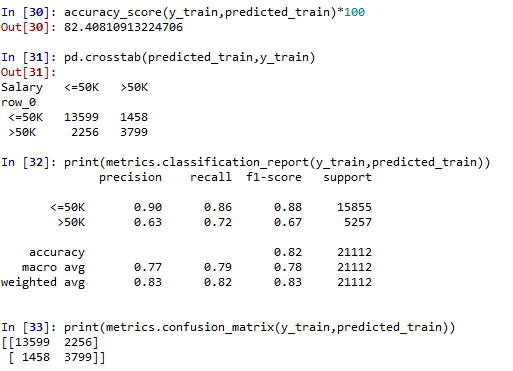
1.) Prepare a classification model using the Naive Bayes algorithm for the salary dataset. Train and test datasets are given separately. Use both for model building. 

**Output:**

**Evaluations on test data**

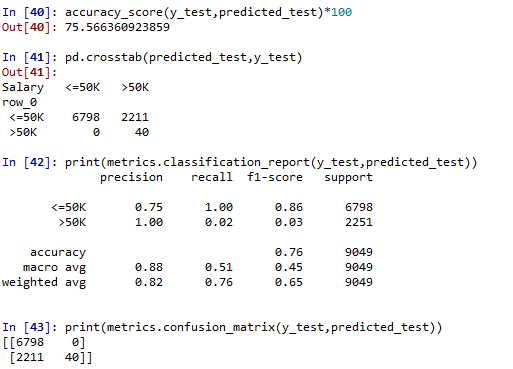
****

**Evaluations on train data**

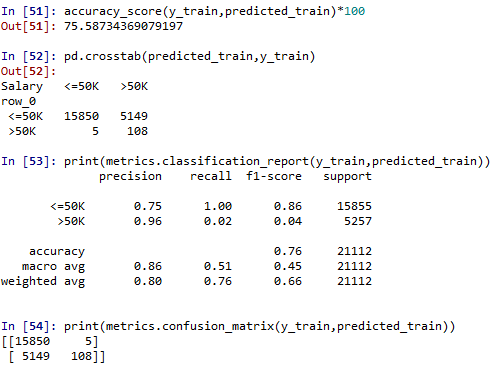
****

**after laplace smoothing:**

**Evaluations on test data**

****

**Evaluations on train data**

****

**PYTHON CODE:**

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer,TfidfTransformer

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

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

from sklearn.metrics import accuracy\_score,confusion\_matrix

#To import dataset

df = pd.read\_csv("C://Users//user//Downloads//naive bayes//SalaryData\_Train.csv",encoding = "ISO-8859-1")

#To find unique values

df['Salary'].unique()

df['relationship'].unique()

df["race"].unique()

df['workclass'].unique()

df['education'].unique()

#To get dummies

df\_new= pd.get\_dummies(df[['workclass', 'education', 'educationno', 'maritalstatus',

'occupation', 'relationship', 'race', 'sex', 'native']],drop\_first=True)

df1=df.drop(['workclass', 'education', 'educationno', 'maritalstatus',

'occupation', 'relationship', 'race', 'sex', 'native','Salary'],axis=1)

df\_y=df.iloc[:,13]

df2=df\_new.iloc[:,0:90]

df\_x= pd.concat([df1,df2],axis=1)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df\_x,df\_y,test\_size=0.30,random\_state=0)

#Gaussian Naive Bayes

model=GaussianNB()

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

#evaluation on test data

predicted\_test=model.predict(x\_test)

predicted\_test

accuracy\_score(y\_test,predicted\_test)\*100

pd.crosstab(predicted\_test,y\_test)

print(metrics.classification\_report(y\_test,predicted\_test))

print(metrics.confusion\_matrix(y\_test,predicted\_test))

#evaluation on train data

predicted\_train=model.predict(x\_train)

predicted\_train

accuracy\_score(y\_train,predicted\_train)\*100

pd.crosstab(predicted\_train,y\_train)

print(metrics.classification\_report(y\_train,predicted\_train))

print(metrics.confusion\_matrix(y\_train,predicted\_train))

#after smoothing guassianNB

#var\_smoothing,default=1e-9

model=GaussianNB(var\_smoothing=3)

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

#evaluation on test data

predicted\_test=model.predict(x\_test)

predicted\_test

accuracy\_score(y\_test,predicted\_test)\*100

pd.crosstab(predicted\_test,y\_test)

print(metrics.classification\_report(y\_test,predicted\_test))

print(metrics.confusion\_matrix(y\_test,predicted\_test))

#evaluation on train data

predicted\_train=model.predict(x\_train)

predicted\_train

accuracy\_score(y\_train,predicted\_train)\*100

pd.crosstab(predicted\_train,y\_train)

print(metrics.classification\_report(y\_train,predicted\_train))

print(metrics.confusion\_matrix(y\_train,predicted\_train))

**Problem Statement: -**

This dataset contains information of users in a social network. This social network has several business clients which can post ads on it. One of the clients has a car company which has just launched a luxury SUV for a ridiculous price. Build a Bernoulli Naïve Bayes model using this dataset and classify which of the users of the social network are going to purchase this luxury SUV. 1 implies that there was a purchase and 0 implies there wasn’t a purchase.

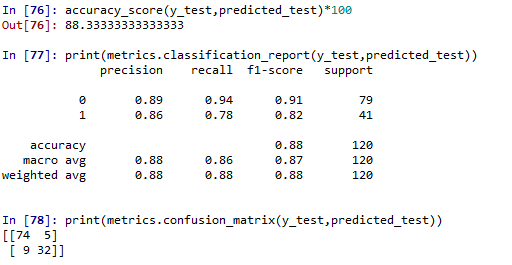
A screenshot of a cell phone

Description automatically generated

**Using GaussianNB** :

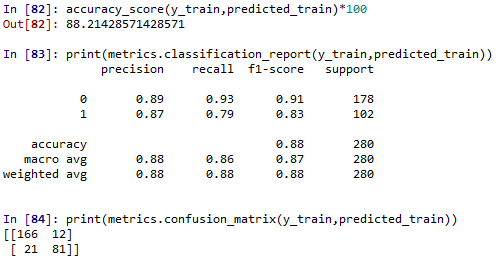
**Evaluations on test data**

**Purchased=1, not purchased=0**



**Evaluations on train data**

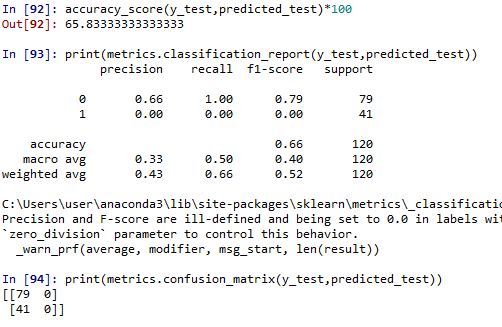
**Purchased=1, not purchased=0**

****

**after laplace smoothing:**

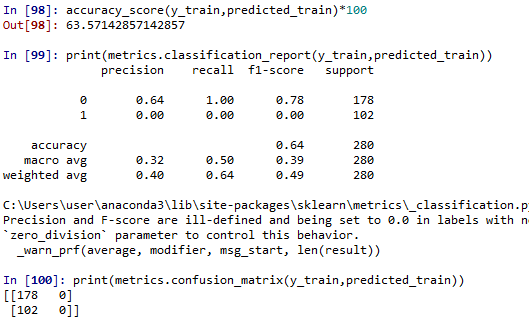
**Evaluations on test data**

**Purchased=1, not purchased=0**

****

**Evaluations on train data**

**Purchased=1, not purchased=0**

****

**PYTHON CODE:**

**import pandas as pd**

**import numpy as np**

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

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

**from sklearn import metrics**

**from sklearn.metrics import accuracy\_score,confusion\_matrix**

**#To import dataset**

**df = pd.read\_csv("C://Users//user//Downloads//naive bayes//NB\_Car\_Ad.csv",encoding = "ISO-8859-1")**

**df.columns**

**gender=pd.get\_dummies(df[['Gender']],drop\_first=True)**

**det = pd.concat([df, gender], join = 'outer', axis = 1)**

**car\_dataX = det.iloc[:, [0,5,2,3]]**

**car\_dataY = det.iloc[:, [4]]**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(car\_dataX,car\_dataY,test\_size=0.30,random\_state=0)**

**#Gaussian Naive Bayes**

**model=GaussianNB()**

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

**#evaluation on test data**

**predicted\_test=model.predict(x\_test)**

**predicted\_test**

**accuracy\_score(y\_test,predicted\_test)\*100**

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

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

**#evaluation on train data**

**predicted\_train=model.predict(x\_train)**

**predicted\_train**

**accuracy\_score(y\_train,predicted\_train)\*100**

**print(metrics.classification\_report(y\_train,predicted\_train))**

**print(metrics.confusion\_matrix(y\_train,predicted\_train))**

**#after smoothing guassianNB**

**#var\_smoothing,default=1e-9**

**model=GaussianNB(var\_smoothing=2)**

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

**#evaluation on test data**

**predicted\_test=model.predict(x\_test)**

**predicted\_test**

**accuracy\_score(y\_test,predicted\_test)\*100**

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

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

**#evaluation on train data**

**predicted\_train=model.predict(x\_train)**

**predicted\_train**

**accuracy\_score(y\_train,predicted\_train)\*100**

**print(metrics.classification\_report(y\_train,predicted\_train))**

**print(metrics.confusion\_matrix(y\_train,predicted\_train))**

**Problem Statement: -**

In this case study, you have been given Twitter data collected from an anonymous twitter handle. With the help of a Naïve Bayes model, predict if a given tweet about a real disaster is real or fake.

**1 = real tweet and 0 = fake tweet**

A screenshot of a cell phone

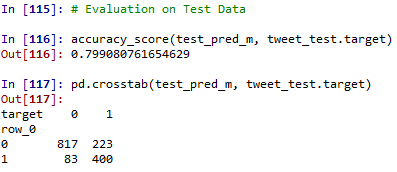
Description automatically generated

**OUTPUT:**

**Using Naive bayes**

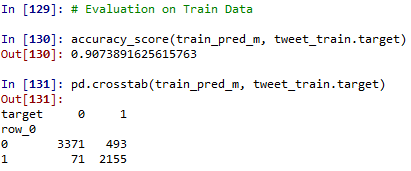
**Evaluations on test data**

**1 = real tweet and 0 = fake tweet**

****

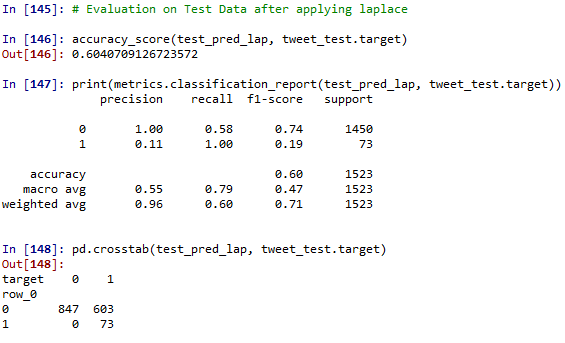
**Evaluations on train data**

**1 = real tweet and 0 = fake tweet**

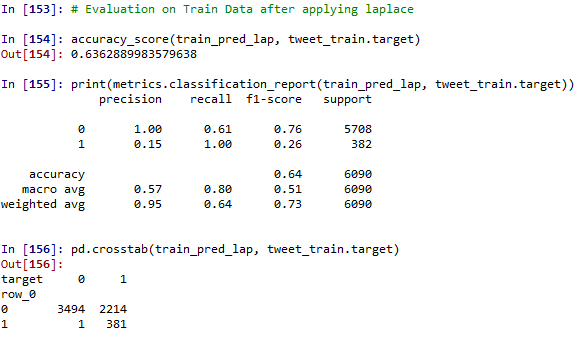
****

**After Laplace smoothing:**

**Evaluations on test data**

****

**Evaluations on train data**

****

**PYTHON CODE:**

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer,TfidfTransformer

from sklearn import metrics

from sklearn.metrics import accuracy\_score,confusion\_matrix

import re

# Loading the data set

tweet\_data = pd.read\_csv("C://Users//user//Downloads//naive bayes//Disaster\_tweets\_NB.csv",encoding = "ISO-8859-1")

# Data cleansing

def cleaning\_text(i):

i = re.sub("[^A-Za-z" "]+"," ",i).lower()

i = re.sub("[0-9" "]+"," ",i)

w = []

for word in i.split(" "):

if len(word)>3:

w.append(word)

return (" ".join(w))

tweet\_data.text = tweet\_data.text.apply(cleaning\_text)

# removing empty rows

tweet\_data = tweet\_data.loc[tweet\_data.text != " ",:]

# CountVectorizer

# Convert a collection of text documents to a matrix of token counts

# splitting data into train and test data sets

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

tweet\_train, tweet\_test = train\_test\_split(tweet\_data, test\_size = 0.2)

# creating a matrix of token counts for the entire text document

def split\_into\_words(i):

return [word for word in i.split(" ")]

# Defining the preparation of email texts into word count matrix format - Bag of Words

tweet\_bow = CountVectorizer(analyzer = split\_into\_words).fit(tweet\_data.text)

# Defining BOW for all messages

all\_tweet\_matrix = tweet\_bow.transform(tweet\_data.text)

# For training messages

train\_tweet\_matrix = tweet\_bow.transform(tweet\_train.text)

# For testing messages

test\_tweet\_matrix = tweet\_bow.transform(tweet\_test.text)

# Learning Term weighting and normalizing on entire emails

tfidf\_transformer = TfidfTransformer().fit(all\_tweet\_matrix)

# Preparing TFIDF for train emails

train\_tfidf = tfidf\_transformer.transform(train\_tweet\_matrix)

train\_tfidf.shape # (row, column)

# Preparing TFIDF for test emails

test\_tfidf = tfidf\_transformer.transform(test\_tweet\_matrix)

test\_tfidf.shape # (row, column)

# Preparing a naive bayes model on training data set

from sklearn.naive\_bayes import MultinomialNB as MB

# Multinomial Naive Bayes

classifier\_mb = MB()

classifier\_mb.fit(train\_tfidf, tweet\_train.target)

# Evaluation on Test Data

test\_pred\_m = classifier\_mb.predict(test\_tfidf)

accuracy\_test\_m = np.mean(test\_pred\_m == tweet\_test.target)

accuracy\_test\_m

from sklearn.metrics import accuracy\_score

# Evaluation on Test Data

accuracy\_score(test\_pred\_m, tweet\_test.target)

pd.crosstab(test\_pred\_m, tweet\_test.target)

# Training Data accuracy

train\_pred\_m = classifier\_mb.predict(train\_tfidf)

accuracy\_train\_m = np.mean(train\_pred\_m == tweet\_train.target)

accuracy\_train\_m

# Evaluation on Train Data

accuracy\_score(train\_pred\_m, tweet\_train.target)

pd.crosstab(train\_pred\_m, tweet\_train.target)

# Multinomial Naive Bayes changing default alpha for laplace smoothing

# if alpha = 0 then no smoothing is applied and the default alpha parameter is 1

# the smoothing process mainly solves the emergence of zero probability problem in the dataset.

classifier\_mb\_lap = MB(alpha = 35)

classifier\_mb\_lap.fit(train\_tfidf, tweet\_train.target)

# Evaluation on Test Data after applying laplace

test\_pred\_lap = classifier\_mb\_lap.predict(test\_tfidf)

accuracy\_test\_lap = np.mean(test\_pred\_lap == tweet\_test.target)

accuracy\_test\_lap

from sklearn.metrics import accuracy\_score

# Evaluation on Test Data after applying laplace

accuracy\_score(test\_pred\_lap, tweet\_test.target)

print(metrics.classification\_report(test\_pred\_lap, tweet\_test.target))

pd.crosstab(test\_pred\_lap, tweet\_test.target)

# Training Data accuracy

train\_pred\_lap = classifier\_mb\_lap.predict(train\_tfidf)

accuracy\_train\_lap = np.mean(train\_pred\_lap == tweet\_train.target)

accuracy\_train\_lap

# Evaluation on Train Data after applying laplace

accuracy\_score(train\_pred\_lap, tweet\_train.target)

print(metrics.classification\_report(train\_pred\_lap, tweet\_train.target))

pd.crosstab(train\_pred\_lap, tweet\_train.target)