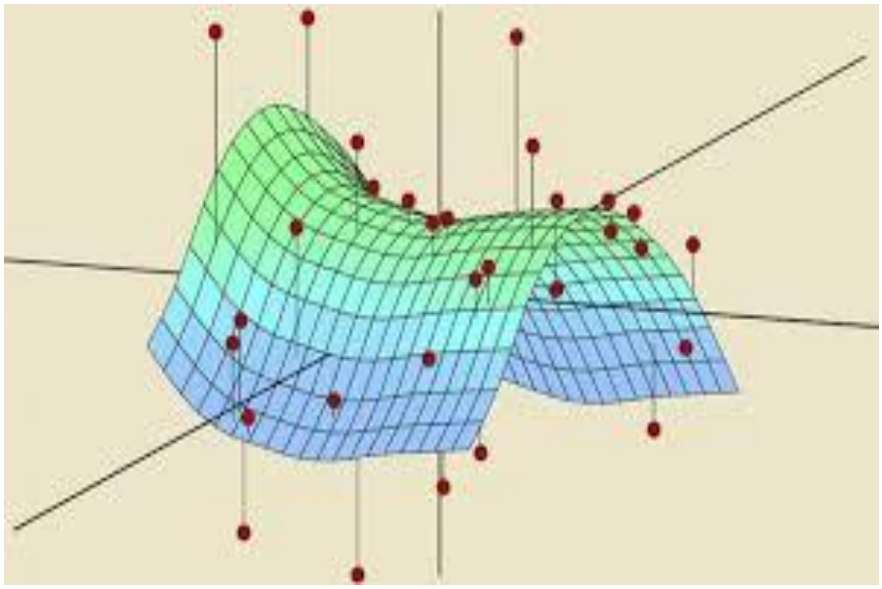
Advance statistics Learning

Project on

Naïve Bayes



Submitted By:

Harshavardhan Perasani

Pravallika pasala

Sujith Reddy Emmadi

Siddharth Dadhich

Naive Bayes

Objective:

• Naive Bayes is a probabilistic technique for creating classifiers in data science. The key assumption of the naive Bayes classifier is that the value of one feature is unrelated to the value of any other feature in the class. Variable

• A Naive Bayes algorithm is a supervised learning technique that uses the Bayes theorem to solve classification problems.

Applications: Where and why do we use them:

• Spam Classification: Given an email, predict whether it is spam or not

• Medical Diagnosis: Given a list of symptoms, predict whether a patient has disease X or not

• Weather: Based on temperature, humidity, etc.... predict if it will rain tomorrow

Methods:

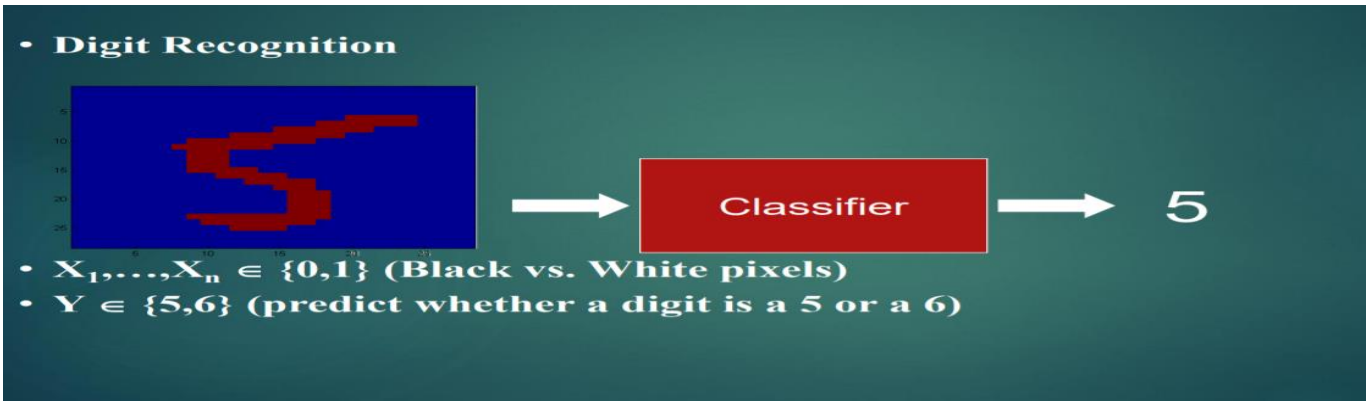
Bayesian Classification

• Problem statement:

• Given features X1, X2,…, Xn

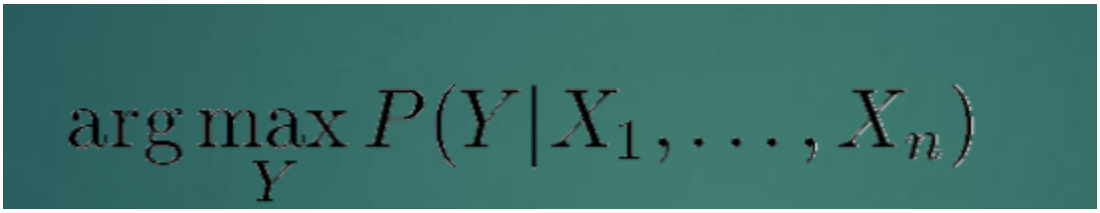
• Predict a label Y

Another Application:



The Bayes Classifier

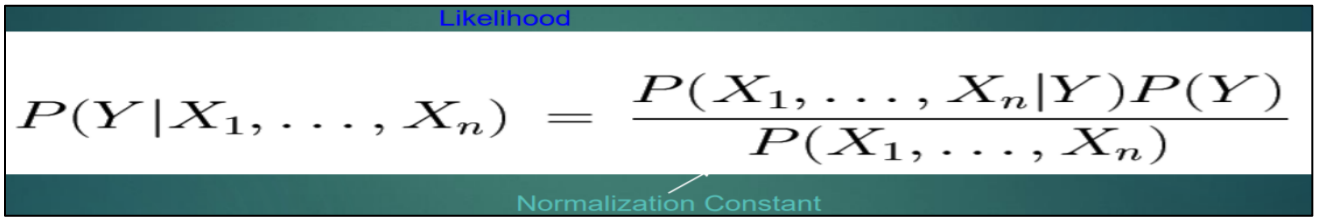
• A good strategy is to predict:



• (for example: what is the probability that the image represents a 5 given its pixels?)

• So … How do we compute that?

• Use Bayes Rule!



Performance of Naïve Bayes in real life dataset:

One of the fastest classification algorithms is Naïve Bayes. It is used in large set of data and various applications like:

1. Text classification
2. Sentiment analysis
3. Recommender systems
4. Spam filtering

Now let us take an example, we will consider a dataset from Kaggle and will use Naïve Bayes in it to make predictions on the salary of workers based on various factors such as age, work class, education qualification etc.. We will then use confusion matrix to find performance of our Naïve Bayes.

Steps:

1. Import dataset:

Dataset used in this section is taken from:

https://www.kaggle.com/wenruliu/adult-income-dataset.

Our dataset has 15 columns and 48,842 data entries.

2. Preprocessing the dataset:

Rename column names

There are few columns which do not have proper column names. So we rename these columns and view our dataset:

Let us look at our dataset after changing names according to our interest and understanding:

```
[5 rows x 15 columns]
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
   age  workclass  fnlwgt  ... hours_per_week  native_country income
0   25    Private  226802  ...             40   United-States  <=50K
1   38    Private   89814  ...             50   United-States  <=50K
2   28  Local-gov  336951  ...             40   United-States   >50K
3   44    Private  160323  ...             40   United-States   >50K
4   18          ?  103497  ...             30   United-States  <=50K

[5 rows x 15 columns]
```

After renaming of columns with meaning, we then try to find if there is any missing value or not.

3. Declare feature vector and target variable:

We declare feature vector and target variable with the help of classification of data by segregating data into numerical and categorical variables. Categorical variables contain object type data, whereas Numerical variable have int type data.

We investigate categorial and numerical variable separately.

There are 9 categorical variables in this dataset:

['workclass', 'education', 'marital_status', occupation', relationship', 'race','sex', native_country' ,'income']

We then find missing values in this dataset and convert them to NaN(Not a number), so we replace all missing values with NaN.

After this we follow same steps for Numerical variables:

['age', 'fnlwgt', 'education_num', 'capital_gain', 'hours_per_week']

Once this process is done, we split data into training and test data.

4. Split data into training and test data set:

It is an important step before we could fit our model into data and make predictions. We use train_test_split library of sklearn package to split the data into training and testing data. Let us see the results after splitting the data:

```
================================
(34189, 14)
(14653, 14)
================================
```

Here 34189 is set of training data and 14653 is test data.

Now we use categorical and numerical variable here as x and y. So, x_train and y_train are for training model and x_test and y_test are for testing model.

5. Model Training:

After splitting of data, we use training data to train our model with Gaussian Naïve Bayes. It is easy to train the data with Naïve Bayes in python with the help of GussianNB library in sklearn package.

Let us look at the code:

```
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, y_train)
```

6. Prediction of result:

We then predict the result using our model:

```
y_pred = gnb.predict(X_test)
print("==========================================")
print(y_pred)
print("==========================================")
```

```
==========================================
['<=50K' '>50K' '>50K' ... '<=50K' '<=50K' '<=50K']
==========================================
```

7. Finally, we calculate performance using confusion matrix:

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

These four outcomes are summarized in a confusion matrix given below:

```
# Print the Confusion Matrix and slice it into four pieces
cm = confusion_matrix(y_test, y_pred)
print("============================================================")
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
print("============================================================")

print(classification_report(y_test, y_pred))
```

Output:

```
Confusion matrix

 [[8992 2146]
 [ 701 2814]]

True Positives(TP) =  8992

True Negatives(TN) =  2814

False Positives(FP) =  2146

False Negatives(FN) =  701
============================================================
              precision    recall  f1-score   support

       <=50K       0.93      0.81      0.86     11138
        >50K       0.57      0.80      0.66      3515

    accuracy                           0.81     14653
   macro avg       0.75      0.80      0.76     14653
weighted avg       0.84      0.81      0.82     14653

======================================================================
```

Understanding the output:

Therefore, our confusion matrix shows:

The confusion matrix shows 5999 + 1897 = 7896 correct predictions and 1408 + 465 = 1873 incorrect predictions.

Finally we calculate accuracy of our model using the formula below:

```
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
```

And we got accuracy of 0.8083 which is equal to 80 %.

Comparison of Naïve Bayes with other models :

we are comparing Naïve Bayes with two types of models namely Logistic Regression and K-Nearest Neighbors of the MLAs using resampling methods like cross validation technique using scikit-learn package of python. And then model fit statistics like accuracy, precision value etc. will be calculated for comparison.

The Data set we used here for application of all the MLAs is diabetes.csv

Which is the data collected by "National Institute of Diabetes and Digestive and Kidney Diseases" contains vital parameters of diabetes patients belong to Pima Indian heritage.

The following is the sample 10 rows of data set

diabetes Pregnancies	diabetes Glucose	diabetes Blood Pressure	diabetes Skin Thickness	diabetes Insulin	diabetes BMI	diabetes Diabetes Pedigree...	diabetes Age	diabetes Outcome
6	148	72	35	0	33.6000	0.62700	50	1
1	85	66	29	0	26.6000	0.35100	31	0
8	183	64	0	0	23.3000	0.67200	32	1
1	89	66	23	94	28.1000	0.16700	21	0
0	137	40	35	168	43.1000	2.28800	33	1
5	116	74	0	0	25.6000	0.20100	30	0
3	78	50	32	88	31.0000	0.24800	26	1
10	115	0	0	0	35.3000	0.13400	29	0
2	197	70	45	543	30.5000	0.15800	53	1
8	125	96	0	0	0.0000	0.23200	54	1

Diabetes data set for ANN

Now, we have implemented a code using python to compare different models.

Step1: First we have to load all the packages needed in this comparison. Besides the basic packages like pandas, numpy, matplotlib we will import some of the scikit-learn packages for application of the MLAs and their comparison.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error,confusion_matrix, precision_score, recall_score, auc,roc_curve
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import svm,model_selection, tree, linear_model, neighbors, naive_bayes, ensemble, discriminant_analysis, g
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
dataset=pd.read_csv('//Users//pravallikapasala//OneDrive - The University of Memphis//Desktop//Prof. Mazjid Noroozi//di
```

Step2: We will load the diabetes data set and check for any null values in the data frame. Checking the data set for any NULL values is very essential, as MLAs can not handle NULL values.so we will eliminate the records with NULL values or replace them with the mean/median of the other values. Then we will store the independent and dependent variables diabetes count using X and Y.

Splitting the data set

Here the data set has been divided into train and test data set. The test data set size is 20% of the total records. This test data will not be used in model training and work as an independent test data.

```
dataset=pd.read_csv('//Users//pravallikapasala//OneDrive - The University of Memphis//Desktop//Prof. Mazjid Noroozi//dia
dataset.isnull().sum()
x=dataset.iloc[:,: -1]
y=dataset.iloc[:,-1]
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.2, random_state=0)
models = []
```

Step3: Storing machine learning algorithms (MLA) in a variable then Creating a box plot to compare there accuracy

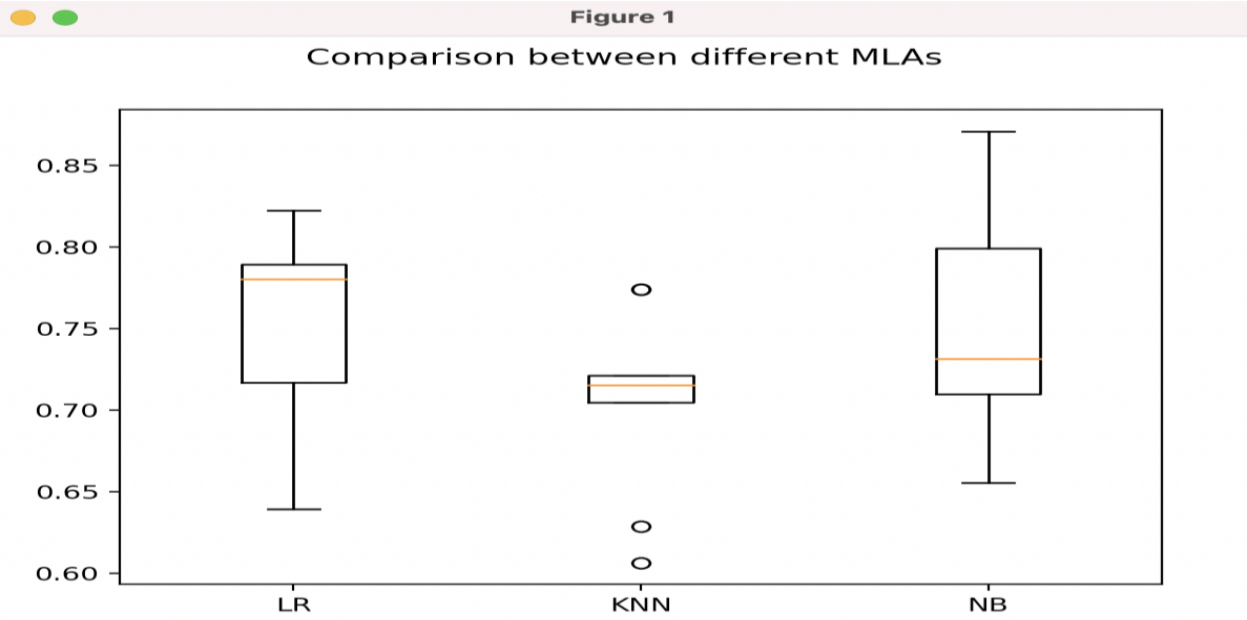
We have selected Logistic Regression, K-Nearest Neighbor models for comparing with our model Naïve Bayes and stored in a variable; so that which can be used at later part of the process. Also, we plotted using boxplot to visualize the models with their cross-validation score.

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
results = []
names = []
scoring = 'accuracy'
seed=42
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, shuffle=True,random_state=seed)
    cv_results = model_selection.cross_val_score(model, x_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Output: The cross-validation scores are printed below and we can see that Logistic Regression and Naïve Bayes analysis are the two most accurate MLAs.

```
LR: 0.755632 (0.058565)
KNN: 0.706742 (0.050821)
NB: 0.747567 (0.064592)
```

Below is the visual representation of the result using box plot



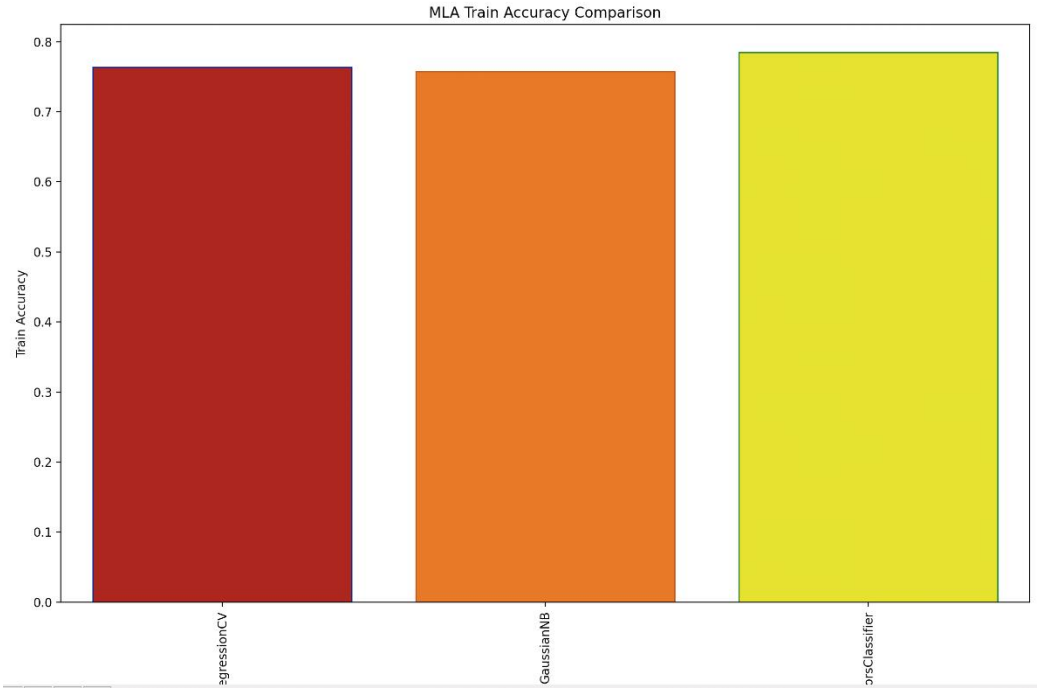
Step4: in the next step we compared all the Machine Learning Algorithms using the Functions in Python.

```
MLA = [
    # GLM
    linear_model.LogisticRegressionCV(),
    naive_bayes.GaussianNB(),
    # Nearest Neighbor
    neighbors.KNeighborsClassifier(),
]
MLA_columns = []
MLA_compare = pd.DataFrame(columns=MLA_columns)

row_index = 0
for alg in MLA:
    predicted = alg.fit(x_train, y_train).predict(x_test)
    fp, tp, th = roc_curve(y_test, predicted)
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index, 'MLA used'] = MLA_name
    MLA_compare.loc[row_index, 'Train Accuracy'] = round(alg.score(x_train, y_train), 4)
    MLA_compare.loc[row_index, 'Test Accuracy'] = round(alg.score(x_test, y_test), 4)
    MLA_compare.loc[row_index, 'Precission'] = precision_score(y_test, predicted)
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_test, predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)
    row_index += 1

MLA_compare.sort_values(by=['Test Accuracy'], ascending=False, inplace=True)
MLA_compare
```

Step5: Then we created a plot to compare train accuracy for three models.

```
# Creating plot to show the train accuracy
plt.subplots(figsize=(13,5))
sns.barplot(x="MLA used", y="Train Accuracy",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('MLA Train Accuracy Comparison')
plt.show()
```

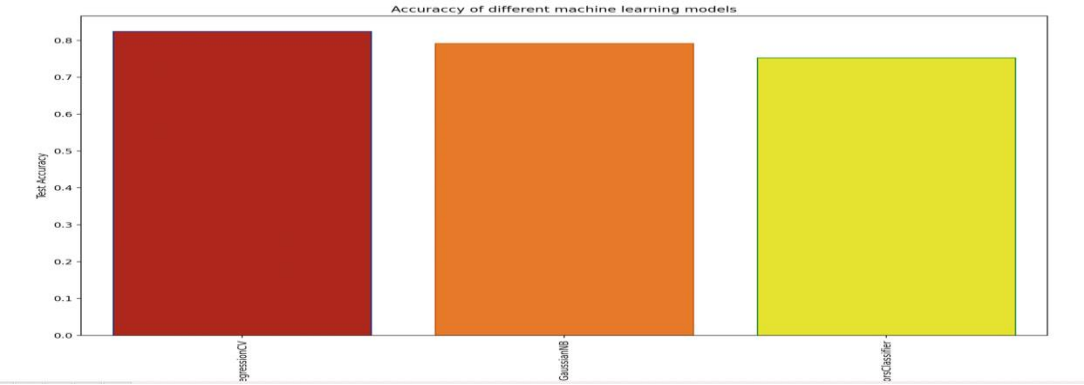
Output:



Step6: Then we created a plot to compare test accuracy for three models.

```
# Creating plot to show the test accuracy
plt.subplots(figsize=(13,5))
sns.barplot(x="MLA used", y="Test Accuracy",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('Accuracy of different machine learning models')
plt.show()
```

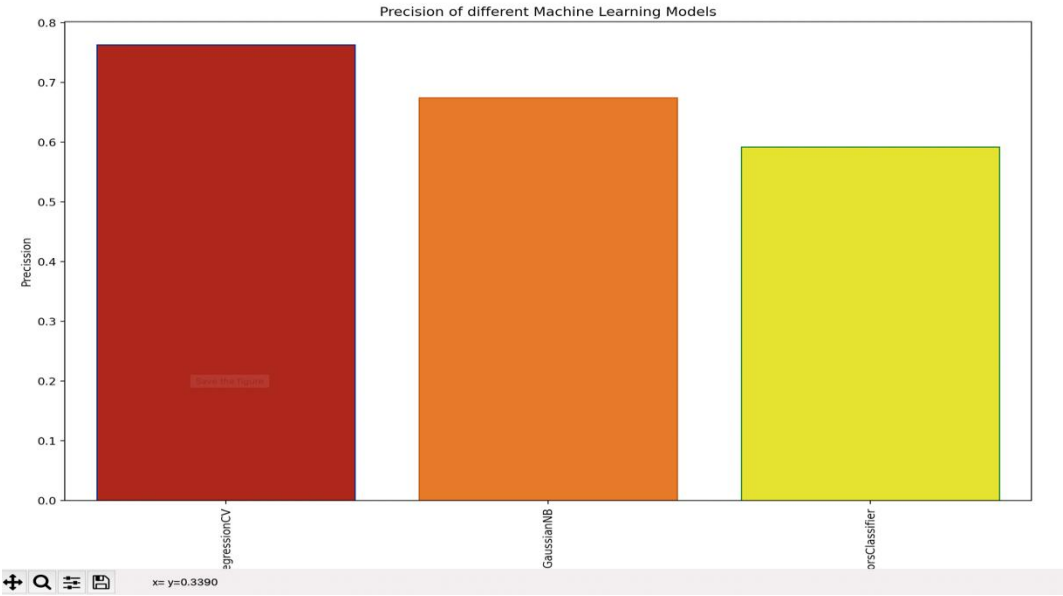
Output:



Step7: Then we created a plot to compare train precision for three models.

```
# Creating plots to compare precission of the MLAs
plt.subplots(figsize=(13,5))
sns.barplot(x="MLA used", y="Precission",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('Precision of different Machine Learning Models')
plt.show()
```

Output:



Conclusion:

By considering all the observations like accuracy, precision we did so far we can conclude that Logistic regression has the highest precision and accuracy whereas Naïve Bayes has second highest Precision and accuracy values compared to K-Nearest Neighbor classification.

Project Submission Form

SUBMITTED BY	SIDDHARTH DADHICH HarshaVardhan Perasani Pravallika Pasala Sujith Reddy Emmadi
UNIVERSITY ID	U00793590 U00827959
COURSE	ADVANCE STAT LEARNING-1 MATH-7635-001
HOMEWORK	1
SUBMITTED TO	DR. MAJID NOROOZI
DATE SUBMITTED	10/13/2021
Due date	10/13/2021