# PROJECT TITLE:-

# PREDICTION OF QUALITY OF WINE USING MACHINE LEARNING ALGORITHMS

**PREPARED BY-**SHASHI NANDAN PRASAD

**INSTITUTE-**NIT JAMSHEDPUR

**BRANCH-**PRODUCTION AND INDUSTRIAL ENGINEERING

**ROLL NO**-2017UGPI015

## CONTENTS

# 1.INTRODUCTION

The aim of this project is to predict the quality of wine on a scale of 3-8 which is the target variable i.e ouputs.The dataset used here is of “Wine Quality Data set” which is picked from UCI Machine Repository.In this dataset, input variables are fixed acidity,volatile acidity,citric acid,residual sugar,chlorides,free sulphates,alcohol.

The ouput variable is quality wine which is scale from 3-8 i.e[ 3,4,5,6,7,8].Higher value represents better quality of wine.

In this paper,I am going to explain the steps I had taken for predicting quality of wine using different Classification techniques and among them which is the best and optimum classification technique for this prediction of quality of wine.

Classification techniques used here are as follows:-

- Xgboost Classifier
- Linear Discriminant Analysis (LDA)
- Logistic Regression using LDA
- Kernel SVM
- K-Nearest Neighbors
- Naïve Bayes Classifier
- Decision Tree Classifier
- Random Forest Classifier

I also used Feature selection Technique to predict the top 5 features related to quality of wine.

## 2.)PROBLEM STATEMENT

Our objective is to predict the quality of wine using various Machine Learning Algorithms by applying various classification Techniques.

## 3.)XGBOOST CLASSIFIER

eXtreme Gradient Boosting or XGBoost is a library of gradient boosting algorithms optimized for modern data science problems and tools. It leverages the techniques mentioned with boosting and comes wrapped in an easy to use library. Some of the major benefits of XGBoost are that its highly scalable/parallelizable, quick to execute, and typically

outperforms other algorithms.

## 4.)LINEAR DISCRIMINANT ANALYSIS(LDA) and LOGISTIC REGRESSION

**Linear discriminant analysis** (**LDA**), **normal discriminant analysis** (**NDA**), or **discriminant function analysis** is a generalization of **Fisher's linear discriminant**, a method used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

## Assumptions:-

- Multivariate normality: Independent variables are normal for each level of the grouping variable.
- Homogeneity of variance/covariance (homoscedasticity): Variances among group variables are the same across levels of predictors. Can be tested with Box's M statistic.[9] It has been suggested, however, that linear discriminant analysis be used when covariances are equal, and that quadratic discriminant analysis may be used when covariances are not equal.
- Multicollinearity: Predictive power can decrease with an increased correlation between predictor variables.
- Independence: Participants are assumed to be randomly sampled, and a participant's score on one variable is assumed to be independent of scores on that variable for all other participants.

In statistics, the **logistic model** (or **logit model**) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, **logistic regression**[1] (or **logit regression**) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

# 5.)KERNEL SVM

In machine learning, **kernel methods** are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified *feature map*: in contrast, kernel methods require only a user-specified *kernel*, i.e.,

a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, *implicit* feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick".[1] Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

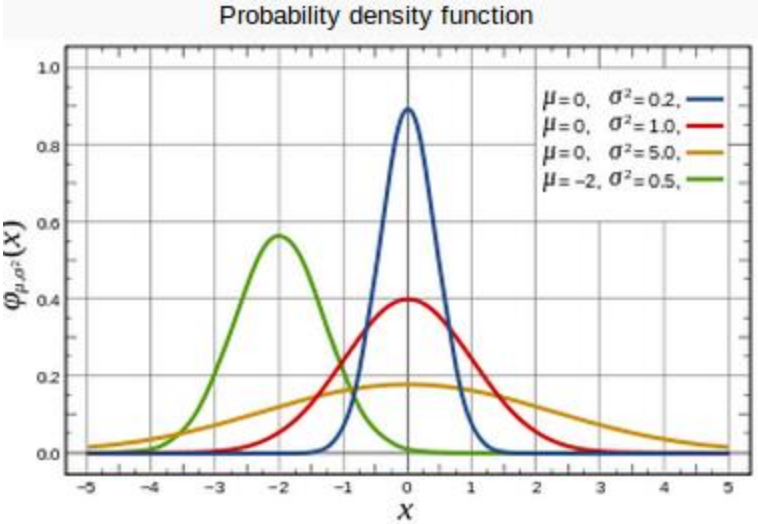
**MATHEMATICAL EXPRESSION :-**

$$k(x, x') = exp(-\frac{||x - x'||^2}{2\sigma^2})$$

**X-Distance from landmark**

**X'-Landmark**

**sigma** is, as usually defined in a Gaussian Distribution, is standard deviation. It determines the width for Gaussian distribution, as shown in the following:-



The red curve is the *standard normal distribution*

# 6.)K-NEAREST NEIGHBORS CLASSIFIER

In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression

**Distance functions**

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise; however, the compromise is that the distinct boundaries within the feature space are blurred.

- In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of *k* nearest neighbors

# 7.)NAiVE BAYES CLASSIFIER

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models.[1] But they could be coupled with Kernel density estimation and achieve higher accuracy levels

# Naive Bayes

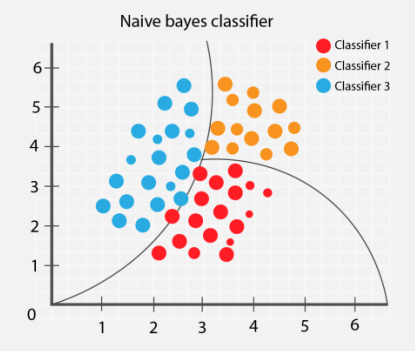
thatware.co

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

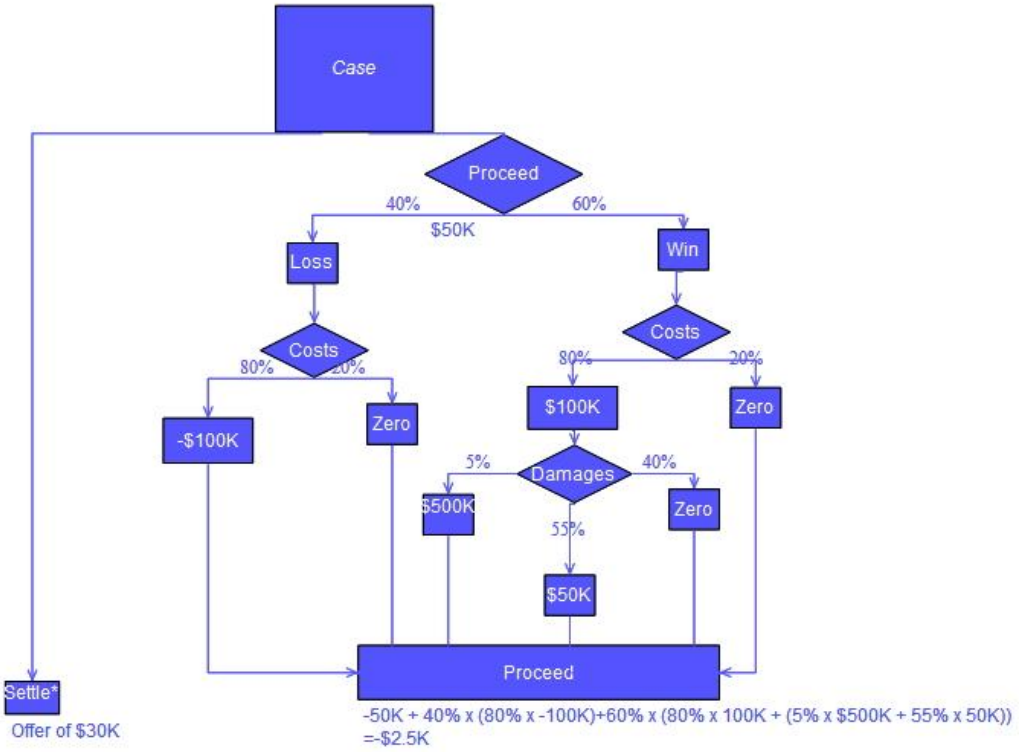
$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$




## 8.)DECISION TREE CLASSIFIER

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.




- Information regarding the dependent variable is done by splitting the dataset of dependent variable using “**Information Entropy**”.
- If at further splitting,it can’t get more information ,there it stops.
- Algorithm finds the optimal split of dataset into these leaves called as “**Terminal Leaves**”.
- We take average of each points of each terminal leaves
- It is a Non-linear,non-continuous classification model
- Suitable for 2D graph Visualisation.

# 9.)RANDOM FOREST CLASSIFIER

**Random forests** or **random decision forests** are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

## ENSEMBLE LEARNING:-

- It refers to collection of Machine Learning algorithms and form a specific ML algorithm termed as “Ensemble Learning”
- n_estimators parameter help to detect overfitting and help the machine to predict better optimum result.

# 10.)MACHINE LEARNING MODEL

For designing a model following steps were taken:-

a) **Dataset Source(Kaggle):-**

https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009

b.) **Data Preparation:** The data was combined in a single csv file for further processing. Our aim was to predict the “**quality**” of wine, so it became our dependent variable. A correlation matrix was checked to find the relation among the parameters. The following variables were considered for model making-

- Fixed acidity
- Volatile acidity

- Citric acid
- Residual sugar
- Chlorides
- Free suphur dioxide
- Total sulphur dioxide
- Density
- PH
- Sulphate
- Alcohol

**c)Choosing a model:** Different models were trained on the train set and its overall accuracy i.e mean_accuracy was predicted by using "**K-FOLD CROSS VALIDATION"** technique:-

**a)Xgboost Classifier**

**Mean_Accuracy=**0.6309860736747529

**b)Logistic Regression using LDA**

**Mean_Accuracy=**0.5773079514824797

**c)Kernel SVM**

**Mean_Accuracy=**0.5924809074573226

**d)K-nearest neighbor classifier**

**Mean_Accuracy=**0.534389038634321 6

**e)Naïve Bayes Classifier**

**Mean_Accuracy=0.5889094788858941**

**f)Decision tree classifier**

**Mean_Accuracy=0.5692834681042228**

**g)Random Forest classifier**

**Mean_Accuracy=0.6300819856244385**

# 11.)RESULT AND CONCLUSION

The best fit is 63.09% for **Xgboost Classifier,** which is quite good as compared to other classifier. Hence, This model can be utilized for the prediction of quality of wine.

# 12.)Annexure

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
df=pd.read_csv('../input/datasets_4458_8204_winequality-red.csv')
df
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7.4 | 0.700 | 0.00 | 1.90 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.400000 | 5 |
| 1 | 7.8 | 0.880 | 0.00 | 2.60 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.800000 | 5 |
| 2 | 7.8 | 0.760 | 0.04 | 2.30 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.800000 | 5 |
| 3 | 11.2 | 0.280 | 0.56 | 1.90 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.800000 | 6 |
| 4 | 7.4 | 0.700 | 0.00 | 1.90 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.400000 | 5 |
| 5 | 7.4 | 0.660 | 0.00 | 1.80 | 0.075 | 13.0 | 40.0 | 0.99780 | 3.51 | 0.56 | 9.400000 | 5 |
| 6 | 7.9 | 0.600 | 0.06 | 1.60 | 0.069 | 15.0 | 59.0 | 0.99640 | 3.30 | 0.46 | 9.400000 | 5 |
| 7 | 7.3 | 0.650 | 0.00 | 1.20 | 0.065 | 15.0 | 21.0 | 0.99460 | 3.39 | 0.47 | 10.000000 | 7 |
| 8 | 7.8 | 0.580 | 0.02 | 2.00 | 0.073 | 9.0 | 18.0 | 0.99680 | 3.36 | 0.57 | 9.500000 | 7 |
| 9 | 7.5 | 0.500 | 0.36 | 6.10 | 0.071 | 17.0 | 102.0 | 0.99780 | 3.35 | 0.80 | 10.500000 | 5 |

#Checking of missing values

df.isnull().sum()

```
fixed acidity         0
volatile acidity      0
citric acid           0
residual sugar        0
chlorides             0
free sulfur dioxide   0
total sulfur dioxide  0
density               0
pH                    0
sulphates             0
alcohol               0
quality               0
dtype: int64
```

# Correlation Matrix

```
corr_matrix = df.corr()
corr_matrix
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fixed acidity | 1.000000 | -0.256131 | 0.671703 | 0.114777 | 0.093705 | -0.153794 | -0.113181 | 0.668047 | -0.682978 | 0.183006 | -0.061668 | 0.124052 |
| volatile acidity | -0.256131 | 1.000000 | -0.552496 | 0.001918 | 0.061298 | -0.010504 | 0.076470 | 0.022026 | 0.234937 | -0.260987 | -0.202288 | -0.390558 |
| citric acid | 0.671703 | -0.552496 | 1.000000 | 0.143577 | 0.203823 | -0.060978 | 0.035533 | 0.364947 | -0.541904 | 0.312770 | 0.109903 | 0.226373 |
| residual sugar | 0.114777 | 0.001918 | 0.143577 | 1.000000 | 0.055610 | 0.187049 | 0.203028 | 0.355283 | -0.085652 | 0.005527 | 0.042075 | 0.013732 |
| chlorides | 0.093705 | 0.061298 | 0.203823 | 0.055610 | 1.000000 | 0.005562 | 0.047400 | 0.200632 | -0.265026 | 0.371260 | -0.221141 | -0.128907 |
| free sulfur dioxide | -0.153794 | -0.010504 | -0.060978 | 0.187049 | 0.005562 | 1.000000 | 0.667666 | -0.021946 | 0.070377 | 0.051658 | -0.069408 | -0.050656 |
| total sulfur dioxide | -0.113181 | 0.076470 | 0.035533 | 0.203028 | 0.047400 | 0.667666 | 1.000000 | 0.071269 | -0.066495 | 0.042947 | -0.205654 | -0.185100 |
| density | 0.668047 | 0.022026 | 0.364947 | 0.355283 | 0.200632 | -0.021946 | 0.071269 | 1.000000 | -0.341699 | 0.148506 | -0.496180 | -0.174919 |
| pH | -0.682978 | 0.234937 | -0.541904 | -0.085652 | -0.265026 | 0.070377 | -0.066495 | -0.341699 | 1.000000 | -0.196648 | 0.205633 | -0.057731 |
| sulphates | 0.183006 | -0.260987 | 0.312770 | 0.005527 | 0.371260 | 0.051658 | 0.042947 | 0.148506 | -0.196648 | 1.000000 | 0.093595 | 0.251397 |
| alcohol | -0.061668 | -0.202288 | 0.109903 | 0.042075 | -0.221141 | -0.069408 | -0.205654 | -0.496180 | 0.205633 | 0.093595 | 1.000000 | 0.476166 |
| quality | 0.124052 | -0.390558 | 0.226373 | 0.013732 | -0.128907 | -0.050656 | -0.185100 | -0.174919 | -0.057731 | 0.251397 | 0.476166 | 1.000000 |

# Feature Selection

```
from sklearn.feature_selection import SelectKBes
t
from sklearn.feature_selection import f_classif
x = df.iloc[:,0:-1]  #independent columns
y = df.iloc[:,-1]    #target column
#apply SelectKBest class to extract top 5 best f
eatures
bestfeatures = SelectKBest(score_func=f_classif,
k=5)
fit = bestfeatures.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],a
xis=1)
featureScores.columns = ['Factors','Score']  #na
ming the dataframe columns
print(featureScores.nlargest(5,'Score'))  #print
5 best features
```

| | Factors | Score |
|---|---|---|
| 10 | alcohol | 115.854797 |
| 1 | volatile acidity | 60.913993 |
| 6 | total sulfur dioxide | 25.478510 |
| 9 | sulphates | 22.273376 |
| 2 | citric acid | 19.690664 |

```
x=df[["alcohol","volatile acidity","sulphates","
citric acid","total sulfur dioxide"]]
y=df[["quality"]]
#splitting the dataset into training set and tes
t set
from sklearn.model_selection import train_test_s
plit
x_train,x_test,y_train,y_test=train_test_split(x
,y,test_size=0.3,random_state=0)
```

## #Xgboost classifier

```
#fitting xgb to the training set
from xgboost import XGBClassifier
classifier=XGBClassifier()
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

## #Linear Discriminant Analysis (LDA)

```
#feature scaling
from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler()
x_train=sc_x.fit_transform(x_train)
x_test=sc_x.transform(x_test)

#applying the LDA
from sklearn.discriminant_analysis import Linear
DiscriminantAnalysis as LDA
lda=LDA(n_components=2)
x_train=lda.fit_transform(x_train,y_train)
x_test=lda.transform(x_test)
```

```
x_train
#two new extracted features that will separate m
ost of the classes of the target variable
```

```
array([[ 1.96813734,  1.61596119],
       [ 2.17784892,  2.49385466],
       [-0.79748674, -0.10435146],
       ...,
       [-1.02754894,  0.60059522],
       [ 1.7622547 ,  0.39600936],
       [-2.95739101, -0.71976609]])
```

```
#APPLYING LOGISTIC REGRESSION ON EXTRACTED FEATU
RES

#fitting the logistic regression to the training
set
from sklearn.linear_model import LogisticRegress
ion
classifier=LogisticRegression(random_state=0)
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

# #KERNEL SVM

```
#fitting the kernel svm to the training dataset
from sklearn.svm import SVC
```

```
classifier=SVC(kernel='rbf',random_state=0)
classifier.fit(x_train,y_train)

#applying grid_search to find optimal model and
optimal parameters
from sklearn.model_selection import GridSearchCV
parameters=[{'C':[1,10,100,1000],'kernel':['line
ar']},
            {'C':[1,10,100,1000],'kernel':['rbf'
],'gamma':[0.5,0.1,0.01,0.001,0.0001]}]

grid_search=GridSearchCV(estimator=classifier,
                         param_grid=parameters,
                         scoring='accuracy',
                         cv=7,
                         n_jobs=-1)

grid_search=grid_search.fit(x_train,y_train)

best_parameter=grid_search.best_params_
best_accuracy=grid_search.best_score_

best_parameter
```

```
{'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
```

```
#fitting the kernel svm to the training dataset
from sklearn.svm import SVC
classifier=SVC(kernel='rbf',C=100,gamma=0.5)
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
```

```
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

## #K-NEAREST NEIGHBOR CLASSIFIER

```
#fitting the k nearest neighbour to the training
set
from sklearn.neighbors import KNeighborsClassifi
er
classifier=KNeighborsClassifier(n_neighbors=5,me
tric= 'minkowski',p=2)
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

## #NAIVE BAYES CLASSIFIER

```
#fitting the naive bayes to the training dataset
from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

## #DECISION TREE CLASSIFIER

```
#fitting the decision tree classifier to the tra
ining set
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='ent
ropy',random_state=0)
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)

#applying the k-for cross validation
#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
```

```
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```

## #RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassif
ier
classifier=RandomForestClassifier(n_estimators=1
00,criterion='entropy',random_state=0)
classifier.fit(x_train,y_train)

#predicting the test set results
y_pred=classifier.predict(x_test)


#best way to find accuracy of model
from sklearn.model_selection import cross_val_sc
ore
accuracy=cross_val_score(estimator=classifier,X=
x_train,y=y_train,cv=7)

accuracy.mean()
```