**Importing required libraries**

In [3]:

**import** numpy **as** np

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

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

**%matplotlib** inline

**import** seaborn **as** sns

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

**import** warnings

warnings**.**filterwarnings('ignore')

**Reading the dataset**

In [5]:

data **=** pd**.**read\_csv(r"dataset\_website.csv")

data**.**head()

Out[5]:

|  | **index** | **having\_IPhaving\_IP\_Address** | **URLURL\_Length** | **Shortining\_Service** | **having\_At\_Symbol** | **double\_slash\_redirecting** | **Prefix\_Suffix** | **having\_Sub\_Domain** | **SSLfinal\_State** | **Domain\_registeration\_length** | **...** | **popUpWidnow** | **Iframe** | **age\_of\_domain** | **DNSRecord** | **web\_traffic** | **Page\_Rank** | **Google\_Index** | **Links\_pointing\_to\_page** | **Statistical\_report** | **Result** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | -1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | ... | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 |
| **1** | 2 | 1 | 1 | 1 | 1 | 1 | -1 | 0 | 1 | -1 | ... | 1 | 1 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | -1 |
| **2** | 3 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | ... | 1 | 1 | 1 | -1 | 1 | -1 | 1 | 0 | -1 | -1 |
| **3** | 4 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | ... | 1 | 1 | -1 | -1 | 1 | -1 | 1 | -1 | 1 | -1 |
| **4** | 5 | 1 | 0 | -1 | 1 | 1 | -1 | 1 | 1 | -1 | ... | -1 | 1 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | 1 |

5 rows × 32 columns

In [6]:

data**.**shape

Out[6]:

(11055, 32)

In [7]:

data**.**columns

Out[7]:

Index(['index', 'having\_IPhaving\_IP\_Address', 'URLURL\_Length',

'Shortining\_Service', 'having\_At\_Symbol', 'double\_slash\_redirecting',

'Prefix\_Suffix', 'having\_Sub\_Domain', 'SSLfinal\_State',

'Domain\_registeration\_length', 'Favicon', 'port', 'HTTPS\_token',

'Request\_URL', 'URL\_of\_Anchor', 'Links\_in\_tags', 'SFH',

'Submitting\_to\_email', 'Abnormal\_URL', 'Redirect', 'on\_mouseover',

'RightClick', 'popUpWidnow', 'Iframe', 'age\_of\_domain', 'DNSRecord',

'web\_traffic', 'Page\_Rank', 'Google\_Index', 'Links\_pointing\_to\_page',

'Statistical\_report', 'Result'],

dtype='object')

**Handling the nul values**

In [8]:

data**.**info()

RangeIndex: 11055 entries, 0 to 11054

Data columns (total 32 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 index 11055 non-null int64

1 having\_IPhaving\_IP\_Address 11055 non-null int64

2 URLURL\_Length 11055 non-null int64

3 Shortining\_Service 11055 non-null int64

4 having\_At\_Symbol 11055 non-null int64

5 double\_slash\_redirecting 11055 non-null int64

6 Prefix\_Suffix 11055 non-null int64

7 having\_Sub\_Domain 11055 non-null int64

8 SSLfinal\_State 11055 non-null int64

9 Domain\_registeration\_length 11055 non-null int64

10 Favicon 11055 non-null int64

11 port 11055 non-null int64

12 HTTPS\_token 11055 non-null int64

13 Request\_URL 11055 non-null int64

14 URL\_of\_Anchor 11055 non-null int64

15 Links\_in\_tags 11055 non-null int64

16 SFH 11055 non-null int64

17 Submitting\_to\_email 11055 non-null int64

18 Abnormal\_URL 11055 non-null int64

19 Redirect 11055 non-null int64

20 on\_mouseover 11055 non-null int64

21 RightClick 11055 non-null int64

22 popUpWidnow 11055 non-null int64

23 Iframe 11055 non-null int64

24 age\_of\_domain 11055 non-null int64

25 DNSRecord 11055 non-null int64

26 web\_traffic 11055 non-null int64

27 Page\_Rank 11055 non-null int64

28 Google\_Index 11055 non-null int64

29 Links\_pointing\_to\_page 11055 non-null int64

30 Statistical\_report 11055 non-null int64

31 Result 11055 non-null int64

dtypes: int64(32)

memory usage: 2.7 MB

In [9]:

data**.**nunique()

Out[9]:

index 11055

having\_IPhaving\_IP\_Address 2

URLURL\_Length 3

Shortining\_Service 2

having\_At\_Symbol 2

double\_slash\_redirecting 2

Prefix\_Suffix 2

having\_Sub\_Domain 3

SSLfinal\_State 3

Domain\_registeration\_length 2

Favicon 2

port 2

HTTPS\_token 2

Request\_URL 2

URL\_of\_Anchor 3

Links\_in\_tags 3

SFH 3

Submitting\_to\_email 2

Abnormal\_URL 2

Redirect 2

on\_mouseover 2

RightClick 2

popUpWidnow 2

Iframe 2

age\_of\_domain 2

DNSRecord 2

web\_traffic 3

Page\_Rank 2

Google\_Index 2

Links\_pointing\_to\_page 3

Statistical\_report 2

Result 2

dtype: int64

In [10]:

data **=** data**.**drop(['index'],axis **=** 1)

In [11]:

data**.**describe()**.**T

Out[11]:

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **having\_IPhaving\_IP\_Address** | 11055.0 | 0.313795 | 0.949534 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |
| **URLURL\_Length** | 11055.0 | -0.633198 | 0.766095 | -1.0 | -1.0 | -1.0 | -1.0 | 1.0 |
| **Shortining\_Service** | 11055.0 | 0.738761 | 0.673998 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **having\_At\_Symbol** | 11055.0 | 0.700588 | 0.713598 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **double\_slash\_redirecting** | 11055.0 | 0.741474 | 0.671011 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Prefix\_Suffix** | 11055.0 | -0.734962 | 0.678139 | -1.0 | -1.0 | -1.0 | -1.0 | 1.0 |
| **having\_Sub\_Domain** | 11055.0 | 0.063953 | 0.817518 | -1.0 | -1.0 | 0.0 | 1.0 | 1.0 |
| **SSLfinal\_State** | 11055.0 | 0.250927 | 0.911892 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |
| **Domain\_registeration\_length** | 11055.0 | -0.336771 | 0.941629 | -1.0 | -1.0 | -1.0 | 1.0 | 1.0 |
| **Favicon** | 11055.0 | 0.628584 | 0.777777 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **port** | 11055.0 | 0.728268 | 0.685324 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **HTTPS\_token** | 11055.0 | 0.675079 | 0.737779 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Request\_URL** | 11055.0 | 0.186793 | 0.982444 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |
| **URL\_of\_Anchor** | 11055.0 | -0.076526 | 0.715138 | -1.0 | -1.0 | 0.0 | 0.0 | 1.0 |
| **Links\_in\_tags** | 11055.0 | -0.118137 | 0.763973 | -1.0 | -1.0 | 0.0 | 0.0 | 1.0 |
| **SFH** | 11055.0 | -0.595749 | 0.759143 | -1.0 | -1.0 | -1.0 | -1.0 | 1.0 |
| **Submitting\_to\_email** | 11055.0 | 0.635640 | 0.772021 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Abnormal\_URL** | 11055.0 | 0.705292 | 0.708949 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Redirect** | 11055.0 | 0.115694 | 0.319872 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **on\_mouseover** | 11055.0 | 0.762099 | 0.647490 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **RightClick** | 11055.0 | 0.913885 | 0.405991 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **popUpWidnow** | 11055.0 | 0.613388 | 0.789818 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Iframe** | 11055.0 | 0.816915 | 0.576784 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **age\_of\_domain** | 11055.0 | 0.061239 | 0.998168 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |
| **DNSRecord** | 11055.0 | 0.377114 | 0.926209 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |
| **web\_traffic** | 11055.0 | 0.287291 | 0.827733 | -1.0 | 0.0 | 1.0 | 1.0 | 1.0 |
| **Page\_Rank** | 11055.0 | -0.483673 | 0.875289 | -1.0 | -1.0 | -1.0 | 1.0 | 1.0 |
| **Google\_Index** | 11055.0 | 0.721574 | 0.692369 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Links\_pointing\_to\_page** | 11055.0 | 0.344007 | 0.569944 | -1.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **Statistical\_report** | 11055.0 | 0.719584 | 0.694437 | -1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| **Result** | 11055.0 | 0.113885 | 0.993539 | -1.0 | -1.0 | 1.0 | 1.0 | 1.0 |

In [12]:

*#Correlation heatmap*

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

sns**.**heatmap(data**.**corr(), annot**=True**)

plt**.**show()

In [13]:

*# Phishing Count in pie chart*

data['Result']**.**value\_counts()**.**plot(kind**=**'pie',autopct**=**'%1.2f%%')

plt**.**title("Phishing Count")

plt**.**show()

**Splitting the dataset into train and test sets**

In [14]:

*# Splitting the dataset into dependant and independant fetature*

X **=** data**.**drop(["Result"],axis **=**1)

y **=** data["Result"]

In [15]:

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.2, random\_state **=** 42)

X\_train**.**shape, y\_train**.**shape, X\_test**.**shape, y\_test**.**shape

Out[15]:

((8844, 30), (8844,), (2211, 30), (2211,))

In [16]:

*# Creating holders to store the model performance results*

ML\_Model **=** []

accuracy **=** []

f1\_score **=** []

recall **=** []

precision **=** []

*#function to call for storing the results*

**def** storeResults(model, a,b,c,d):

ML\_Model**.**append(model)

accuracy**.**append(round(a, 3))

f1\_score**.**append(round(b, 3))

recall**.**append(round(c, 3))

precision**.**append(round(d, 3))

In [17]:

*# Linear regression model*

**from** sklearn.linear\_model **import** LogisticRegression

log **=** LogisticRegression()

*# fit the model*

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

Out[17]:

LogisticRegression()

In [18]:

*#predicting the target value from the model for the samples*

y\_train\_log **=** log**.**predict(X\_train)

y\_test\_log **=** log**.**predict(X\_test)

In [19]:

*#computing the accuracy, f1\_score, Recall, precision of the model performance*

acc\_train\_log **=** metrics**.**accuracy\_score(y\_train,y\_train\_log)

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

print("Logistic Regression : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_log))

print("Logistic Regression : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_log))

print()

f1\_score\_train\_log **=** metrics**.**f1\_score(y\_train,y\_train\_log)

f1\_score\_test\_log **=** metrics**.**f1\_score(y\_test,y\_test\_log)

print("Logistic Regression : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_log))

print("Logistic Regression : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_log))

print()

recall\_score\_train\_log **=** metrics**.**recall\_score(y\_train,y\_train\_log)

recall\_score\_test\_log **=** metrics**.**recall\_score(y\_test,y\_test\_log)

print("Logistic Regression : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_log))

print("Logistic Regression : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_log))

print()

precision\_score\_train\_log **=** metrics**.**precision\_score(y\_train,y\_train\_log)

precision\_score\_test\_log **=** metrics**.**precision\_score(y\_test,y\_test\_log)

print("Logistic Regression : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_log))

print("Logistic Regression : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_log))

Logistic Regression : Accuracy on training Data: 0.929

Logistic Regression : Accuracy on test Data: 0.924

Logistic Regression : f1\_score on training Data: 0.937

Logistic Regression : f1\_score on test Data: 0.933

Logistic Regression : Recall on training Data: 0.947

Logistic Regression : Recall on test Data: 0.939

Logistic Regression : precision on training Data: 0.927

Logistic Regression : precision on test Data: 0.928

In [20]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.92 0.90 0.91 956

1 0.93 0.94 0.93 1255

accuracy 0.92 2211

macro avg 0.92 0.92 0.92 2211

weighted avg 0.92 0.92 0.92 2211

In [21]:

*#storing the results. The below mentioned order of parameter passing is important.*

storeResults('Logistic Regression',acc\_test\_log,f1\_score\_test\_log,

recall\_score\_train\_log,precision\_score\_train\_log)

In [22]:

*# K-Nearest Neighbors Classifier model*

**from** sklearn.neighbors **import** KNeighborsClassifier

*# instantiate the model*

knn **=** KNeighborsClassifier(n\_neighbors**=**2)

*# fit the model*

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

Out[22]:

KNeighborsClassifier(n\_neighbors=2)

In [23]:

*#predicting the target value from the model for the samples*

y\_train\_knn **=** knn**.**predict(X\_train)

y\_test\_knn **=** knn**.**predict(X\_test)

In [24]:

*#computing the accuracy,f1\_score,Recall,precision of the model performance*

acc\_train\_knn **=** metrics**.**accuracy\_score(y\_train,y\_train\_knn)

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

print("K-Nearest Neighbors : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_knn))

print("K-Nearest Neighbors : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_knn))

print()

f1\_score\_train\_knn **=** metrics**.**f1\_score(y\_train,y\_train\_knn)

f1\_score\_test\_knn **=** metrics**.**f1\_score(y\_test,y\_test\_knn)

print("K-Nearest Neighbors : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_knn))

print("K-Nearest Neighbors : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_knn))

print()

recall\_score\_train\_knn **=** metrics**.**recall\_score(y\_train,y\_train\_knn)

recall\_score\_test\_knn **=** metrics**.**recall\_score(y\_test,y\_test\_knn)

print("K-Nearest Neighborsn : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_knn))

print("Logistic Regression : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_knn))

print()

precision\_score\_train\_knn **=** metrics**.**precision\_score(y\_train,y\_train\_knn)

precision\_score\_test\_knn **=** metrics**.**precision\_score(y\_test,y\_test\_knn)

print("K-Nearest Neighbors : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_knn))

print("K-Nearest Neighbors : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_knn))

K-Nearest Neighbors : Accuracy on training Data: 0.977

K-Nearest Neighbors : Accuracy on test Data: 0.944

K-Nearest Neighbors : f1\_score on training Data: 0.979

K-Nearest Neighbors : f1\_score on test Data: 0.950

K-Nearest Neighborsn : Recall on training Data: 0.962

Logistic Regression : Recall on test Data: 0.929

K-Nearest Neighbors : precision on training Data: 0.996

K-Nearest Neighbors : precision on test Data: 0.971

In [25]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.91 0.96 0.94 956

1 0.97 0.93 0.95 1255

accuracy 0.94 2211

macro avg 0.94 0.95 0.94 2211

weighted avg 0.95 0.94 0.94 2211

In [48]:

training\_accuracy **=** [ ]

test\_accuracy **=** [ ]

*# try max\_depth from 1 to 20*

depth **=** range(1,20)

**for** n **in** depth:

knn **=** KNeighborsClassifier(n\_neighbors**=**n)

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

*# record training set accuracy*

training\_accuracy**.**append(knn**.**score(X\_train, y\_train))

*# record generalization accuracy*

test\_accuracy**.**append(knn**.**score(X\_test, y\_test))

*#plotting the training & testing accuracy for n\_estimators from 1 to 20*

plt**.**plot(depth, training\_accuracy, label**=**"training accuracy")

plt**.**plot(depth, test\_accuracy, label**=**"test accuracy")

plt**.**ylabel("Accuracy")

plt**.**xlabel("n\_neighbors")

plt**.**legend();

In [49]:

*#storing the results. The below mentioned order of parameter passing is important.*

storeResults('K-Nearest Neighbors',acc\_test\_knn,f1\_score\_test\_knn,

recall\_score\_train\_knn,precision\_score\_train\_knn)

In [28]:

*# Support Vector Classifier model*

**from** sklearn.svm **import** SVC

**from** sklearn.model\_selection **import** GridSearchCV

*# defining parameter range*

param\_grid **=** {'gamma': [0.1],'kernel': ['rbf','linear']}

svc **=** GridSearchCV(SVC(), param\_grid)

*# fitting the model for grid search*

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

Out[28]:

GridSearchCV(estimator=SVC(),

param\_grid={'gamma': [0.1], 'kernel': ['rbf', 'linear']})

In [29]:

*#predicting the target value from the model for the samples*

y\_train\_svc **=** svc**.**predict(X\_train)

y\_test\_svc **=** svc**.**predict(X\_test)

In [30]:

*#computing the accuracy, f1\_score, Recall, precision of the model performance*

acc\_train\_svc **=** metrics**.**accuracy\_score(y\_train,y\_train\_svc)

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

print("Support Vector Machine : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_svc))

print("Support Vector Machine : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_svc))

print()

f1\_score\_train\_svc **=** metrics**.**f1\_score(y\_train,y\_train\_svc)

f1\_score\_test\_svc **=** metrics**.**f1\_score(y\_test,y\_test\_svc)

print("Support Vector Machine : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_svc))

print("Support Vector Machine : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_svc))

print()

recall\_score\_train\_svc **=** metrics**.**recall\_score(y\_train,y\_train\_svc)

recall\_score\_test\_svc **=** metrics**.**recall\_score(y\_test,y\_test\_svc)

print("Support Vector Machine : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_svc))

print("Support Vector Machine : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_svc))

print()

precision\_score\_train\_svc **=** metrics**.**precision\_score(y\_train,y\_train\_svc)

precision\_score\_test\_svc **=** metrics**.**precision\_score(y\_test,y\_test\_svc)

print("Support Vector Machine : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_svc))

print("Support Vector Machine : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_svc))

Support Vector Machine : Accuracy on training Data: 0.971

Support Vector Machine : Accuracy on test Data: 0.957

Support Vector Machine : f1\_score on training Data: 0.974

Support Vector Machine : f1\_score on test Data: 0.963

Support Vector Machine : Recall on training Data: 0.982

Support Vector Machine : Recall on test Data: 0.973

Support Vector Machine : precision on training Data: 0.966

Support Vector Machine : precision on test Data: 0.952

In [31]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.96 0.94 0.95 956

1 0.95 0.97 0.96 1255

accuracy 0.96 2211

macro avg 0.96 0.95 0.96 2211

weighted avg 0.96 0.96 0.96 2211

In [32]:

*#storing the results. The below mentioned order of parameter passing is important.*

storeResults('Support Vector Machine',acc\_test\_svc,f1\_score\_test\_svc,

recall\_score\_train\_svc,precision\_score\_train\_svc)

In [33]:

*# Naive Bayes Classifier Model*

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

**from** sklearn.pipeline **import** Pipeline

*# instantiate the model*

nb**=** GaussianNB()

*# fit the model*

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

Out[33]:

GaussianNB()

In [34]:

*#predicting the target value from the model for the samples*

y\_train\_nb **=** nb**.**predict(X\_train)

y\_test\_nb **=** nb**.**predict(X\_test)

In [35]:

*#computing the accuracy, f1\_score, Recall, precision of the model performance*

acc\_train\_nb **=** metrics**.**accuracy\_score(y\_train,y\_train\_nb)

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

print("Naive Bayes Classifier : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_nb))

print("Naive Bayes Classifier : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_nb))

print()

f1\_score\_train\_nb **=** metrics**.**f1\_score(y\_train,y\_train\_nb)

f1\_score\_test\_nb **=** metrics**.**f1\_score(y\_test,y\_test\_nb)

print("Naive Bayes Classifier : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_nb))

print("Naive Bayes Classifier : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_nb))

print()

recall\_score\_train\_nb **=** metrics**.**recall\_score(y\_train,y\_train\_nb)

recall\_score\_test\_nb **=** metrics**.**recall\_score(y\_test,y\_test\_nb)

print("Naive Bayes Classifier : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_nb))

print("Naive Bayes Classifier : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_nb))

print()

precision\_score\_train\_nb **=** metrics**.**precision\_score(y\_train,y\_train\_nb)

precision\_score\_test\_nb **=** metrics**.**precision\_score(y\_test,y\_test\_nb)

print("Naive Bayes Classifier : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_nb))

print("Naive Bayes Classifier : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_nb))

Naive Bayes Classifier : Accuracy on training Data: 0.607

Naive Bayes Classifier : Accuracy on test Data: 0.583

Naive Bayes Classifier : f1\_score on training Data: 0.451

Naive Bayes Classifier : f1\_score on test Data: 0.420

Naive Bayes Classifier : Recall on training Data: 0.291

Naive Bayes Classifier : Recall on test Data: 0.266

Naive Bayes Classifier : precision on training Data: 0.996

Naive Bayes Classifier : precision on test Data: 0.997

In [36]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.96 0.94 0.95 956

1 0.95 0.97 0.96 1255

accuracy 0.96 2211

macro avg 0.96 0.95 0.96 2211

weighted avg 0.96 0.96 0.96 2211

In [37]:

*#storing the results. The below mentioned order of parameter passing is important.*

storeResults('Naive Bayes Classifier',acc\_test\_nb,f1\_score\_test\_nb,

recall\_score\_train\_nb,precision\_score\_train\_nb)

In [38]:

*# Decision Tree Classifier model*

**from** sklearn.tree **import** DecisionTreeClassifier

*# instantiate the model*

tree **=** DecisionTreeClassifier(max\_depth**=**30)

*# fit the model*

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

Out[38]:

DecisionTreeClassifier(max\_depth=30)

In [39]:

*#predicting the target value from the model for the samples*

y\_train\_tree **=** tree**.**predict(X\_train)

y\_test\_tree **=** tree**.**predict(X\_test)

In [40]:

*#computing the accuracy, f1\_score, Recall, precision of the model performance*

acc\_train\_tree **=** metrics**.**accuracy\_score(y\_train,y\_train\_tree)

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

print("Decision Tree : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_tree))

print("Decision Tree : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_tree))

print()

f1\_score\_train\_tree **=** metrics**.**f1\_score(y\_train,y\_train\_tree)

f1\_score\_test\_tree **=** metrics**.**f1\_score(y\_test,y\_test\_tree)

print("Decision Tree : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_tree))

print("Decision Tree : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_tree))

print()

recall\_score\_train\_tree **=** metrics**.**recall\_score(y\_train,y\_train\_tree)

recall\_score\_test\_tree **=** metrics**.**recall\_score(y\_test,y\_test\_tree)

print("Decision Tree : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_tree))

print("Decision Tree : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_tree))

print()

precision\_score\_train\_tree **=** metrics**.**precision\_score(y\_train,y\_train\_tree)

precision\_score\_test\_tree **=** metrics**.**precision\_score(y\_test,y\_test\_tree)

print("Decision Tree : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_tree))

print("Decision Tree : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_tree))

Decision Tree : Accuracy on training Data: 0.991

Decision Tree : Accuracy on test Data: 0.957

Decision Tree : f1\_score on training Data: 0.992

Decision Tree : f1\_score on test Data: 0.963

Decision Tree : Recall on training Data: 0.992

Decision Tree : Recall on test Data: 0.964

Decision Tree : precision on training Data: 0.991

Decision Tree : precision on test Data: 0.961

In [41]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.95 0.95 0.95 956

1 0.96 0.96 0.96 1255

accuracy 0.96 2211

macro avg 0.96 0.96 0.96 2211

weighted avg 0.96 0.96 0.96 2211

In [42]:

training\_accuracy **=** []

test\_accuracy **=** []

*# try max\_depth from 1 to 30*

depth **=** range(1,30)

**for** n **in** depth:

tree\_test **=** DecisionTreeClassifier(max\_depth**=**n)

tree\_test**.**fit(X\_train, y\_train)

*# record training set accuracy*

training\_accuracy**.**append(tree\_test**.**score(X\_train, y\_train))

*# record generalization accuracy*

test\_accuracy**.**append(tree\_test**.**score(X\_test, y\_test))

*#plotting the training & testing accuracy for max\_depth from 1 to 30*

plt**.**plot(depth, training\_accuracy, label**=**"training accuracy")

plt**.**plot(depth, test\_accuracy, label**=**"test accuracy")

plt**.**ylabel("Accuracy")

plt**.**xlabel("max\_depth")

plt**.**legend();

In [55]:

*#storing the results. The below mentioned order of parameter passing is important.*

storeResults('Decision Tree',acc\_test\_tree,f1\_score\_test\_tree,

recall\_score\_train\_tree,precision\_score\_train\_tree)

In [56]:

*# Random Forest Classifier Model*

**from** sklearn.ensemble **import** RandomForestClassifier

*# instantiate the model*

forest **=** RandomForestClassifier(n\_estimators**=**10)

*# fit the model*

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

Out[56]:

RandomForestClassifier(n\_estimators=10)

In [57]:

*#predicting the target value from the model for the samples*

y\_train\_forest **=** forest**.**predict(X\_train)

y\_test\_forest **=** forest**.**predict(X\_test)

In [58]:

*#computing the accuracy, f1\_score, Recall, precision of the model performance*

acc\_train\_forest **=** metrics**.**accuracy\_score(y\_train,y\_train\_forest)

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

print("Random Forest : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_forest))

print("Random Forest : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_forest))

print()

f1\_score\_train\_forest **=** metrics**.**f1\_score(y\_train,y\_train\_forest)

f1\_score\_test\_forest **=** metrics**.**f1\_score(y\_test,y\_test\_forest)

print("Random Forest : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_forest))

print("Random Forest : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_forest))

print()

recall\_score\_train\_forest **=** metrics**.**recall\_score(y\_train,y\_train\_forest)

recall\_score\_test\_forest **=** metrics**.**recall\_score(y\_test,y\_test\_forest)

print("Random Forest : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_forest))

print("Random Forest : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_forest))

print()

precision\_score\_train\_forest **=** metrics**.**precision\_score(y\_train,y\_train\_forest)

precision\_score\_test\_forest **=** metrics**.**precision\_score(y\_test,y\_test\_tree)

print("Random Forest : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_forest))

print("Random Forest : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_forest))

Random Forest : Accuracy on training Data: 0.990

Random Forest : Accuracy on test Data: 0.966

Random Forest : f1\_score on training Data: 0.991

Random Forest : f1\_score on test Data: 0.970

Random Forest : Recall on training Data: 0.992

Random Forest : Recall on test Data: 0.974

Random Forest : precision on training Data: 0.989

Random Forest : precision on test Data: 0.961

In [59]:

*#computing the classification report of the model*

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

precision recall f1-score support

-1 0.97 0.96 0.96 956

1 0.97 0.97 0.97 1255

accuracy 0.97 2211

macro avg 0.97 0.96 0.97 2211

weighted avg 0.97 0.97 0.97 2211

In [60]:

training\_accuracy **=** []

test\_accuracy **=** []

*# try max\_depth from 1 to 20*

depth **=** range(1,20)

**for** n **in** depth:

rfc\_test **=** RandomForestClassifier(n\_estimators**=**n)

rfc\_test**.**fit(X\_train, y\_train)

*# record training set accuracy*

training\_accuracy**.**append(rfc\_test**.**score(X\_train, y\_train))

*# record generalization accuracy*

test\_accuracy**.**append(rfc\_test**.**score(X\_test, y\_test))

*#plotting the training & testing accuracy for n\_estimators from 1 to 20*

plt**.**figure(figsize**=None**)

plt**.**plot(depth, training\_accuracy, label**=**"training accuracy")

plt**.**plot(depth, test\_accuracy, label**=**"test accuracy")

plt**.**ylabel("Accuracy")

plt**.**xlabel("n\_estimators")

plt**.**legend();

In [61]:

storeResults('Random Forest',acc\_test\_rfc,f1\_score\_test\_rfc,

recall\_score\_train\_rfc,precision\_score\_train\_rfc)

**---------------------------------------------------------------------------**

**NameError** Traceback (most recent call last)

in

**----> 1 storeResults('Random Forest',acc\_test\_rfc,f1\_score\_test\_rfc,**

2 recall\_score\_train\_rfc,precision\_score\_train\_rfc)

**NameError**: name 'acc\_test\_rfc' is not defined

In [52]:

**from** sklearn.ensemble **import** GradientBoostingClassifier

*# instantiate the model*

gbc **=** GradientBoostingClassifier(max\_depth**=**4,learning\_rate**=**0.7)

*# fit the model*

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

Out[52]:

GradientBoostingClassifier(learning\_rate=0.7, max\_depth=4)

In [62]:

y\_train\_gbc **=** gbc**.**predict(X\_train)

y\_test\_gbc **=** gbc**.**predict(X\_test)

In [63]:

acc\_train\_gbc **=** metrics**.**accuracy\_score(y\_train,y\_train\_gbc)

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

print("Gradient Boosting Classifier : Accuracy on training Data: {:.3f}"**.**format(acc\_train\_gbc))

print("Gradient Boosting Classifier : Accuracy on test Data: {:.3f}"**.**format(acc\_test\_gbc))

print()

f1\_score\_train\_gbc **=** metrics**.**f1\_score(y\_train,y\_train\_gbc)

f1\_score\_test\_gbc **=** metrics**.**f1\_score(y\_test,y\_test\_gbc)

print("Gradient Boosting Classifier : f1\_score on training Data: {:.3f}"**.**format(f1\_score\_train\_gbc))

print("Gradient Boosting Classifier : f1\_score on test Data: {:.3f}"**.**format(f1\_score\_test\_gbc))

print()

recall\_score\_train\_gbc **=** metrics**.**recall\_score(y\_train,y\_train\_gbc)

recall\_score\_test\_gbc **=** metrics**.**recall\_score(y\_test,y\_test\_gbc)

print("Gradient Boosting Classifier : Recall on training Data: {:.3f}"**.**format(recall\_score\_train\_gbc))

print("Gradient Boosting Classifier : Recall on test Data: {:.3f}"**.**format(recall\_score\_test\_gbc))

print()

precision\_score\_train\_gbc **=** metrics**.**precision\_score(y\_train,y\_train\_gbc)

precision\_score\_test\_gbc **=** metrics**.**precision\_score(y\_test,y\_test\_gbc)

print("Gradient Boosting Classifier : precision on training Data: {:.3f}"**.**format(precision\_score\_train\_gbc))

print("Gradient Boosting Classifier : precision on test Data: {:.3f}"**.**format(precision\_score\_test\_gbc))

Gradient Boosting Classifier : Accuracy on training Data: 0.987

Gradient Boosting Classifier : Accuracy on test Data: 0.971

Gradient Boosting Classifier : f1\_score on training Data: 0.989

Gradient Boosting Classifier : f1\_score on test Data: 0.975

Gradient Boosting Classifier : Recall on training Data: 0.992

Gradient Boosting Classifier : Recall on test Data: 0.988

Gradient Boosting Classifier : precision on training Data: 0.985

Gradient Boosting Classifier : precision on test Data: 0.962

In [64]:

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

precision recall f1-score support

-1 0.98 0.95 0.97 956

1 0.96 0.99 0.97 1255

accuracy 0.97 2211

macro avg 0.97 0.97 0.97 2211

weighted avg 0.97 0.97 0.97 2211

In [65]:

training\_accuracy **=** []

test\_accuracy **=** []

*# try learning\_rate from 0.1 to 0.9*

depth **=** range(1,10)

**for** n **in** depth:

forest\_test **=** GradientBoostingClassifier(learning\_rate **=** n**\***0.1)

forest\_test**.**fit(X\_train, y\_train)

*# record training set accuracy*

training\_accuracy**.**append(forest\_test**.**score(X\_train, y\_train))

*# record generalization accuracy*

test\_accuracy**.**append(forest\_test**.**score(X\_test, y\_test))

In [66]:

plt**.**figure(figsize**=None**)

plt**.**plot(depth, training\_accuracy, label**=**"training accuracy")

plt**.**plot(depth, test\_accuracy, label**=**"test accuracy")

plt**.**ylabel("Accuracy")

plt**.**xlabel("learning\_rate")

plt**.**legend();

In [68]:

training\_accuracy **=** []

test\_accuracy **=** []

*# try learning\_rate from 0.1 to 0.9*

depth **=** range(1,10,1)

**for** n **in** depth:

forest\_test **=** GradientBoostingClassifier(max\_depth**=**n,learning\_rate **=** 0.7)

forest\_test**.**fit(X\_train, y\_train)

*# record training set accuracy*

training\_accuracy**.**append(forest\_test**.**score(X\_train, y\_train))

*# record generalization accuracy*

test\_accuracy**.**append(forest\_test**.**score(X\_test, y\_test))

*#plotting the training & testing accuracy for n\_estimators from 1 to 50*

plt**.**figure(figsize**=None**)

plt**.**plot(depth, training\_accuracy, label**=**"training accuracy")

plt**.**plot(depth, test\_accuracy, label**=**"test accuracy")

plt**.**ylabel("Accuracy")

plt**.**xlabel("max\_depth")

plt**.**legend();

In [69]:

result **=** pd**.**DataFrame({ 'ML Model' : ML\_Model,

'Accuracy' : accuracy,

'f1\_score' : f1\_score,

'Recall' : recall,

'Precision': precision,

})

In [70]:

result

Out[70]:

|  | **ML Model** | **Accuracy** | **f1\_score** | **Recall** | **Precision** |
| --- | --- | --- | --- | --- | --- |
| **0** | Logistic Regression | 0.924 | 0.933 | 0.947 | 0.927 |
| **1** | K-Nearest Neighbors | 0.944 | 0.950 | 0.962 | 0.996 |
| **2** | Support Vector Machine | 0.957 | 0.963 | 0.982 | 0.966 |
| **3** | Naive Bayes Classifier | 0.583 | 0.420 | 0.291 | 0.996 |
| **4** | Decision Tree | 0.957 | 0.963 | 0.992 | 0.991 |
| **5** | K-Nearest Neighbors | 0.944 | 0.950 | 0.962 | 0.996 |
| **6** | Decision Tree | 0.957 | 0.963 | 0.992 | 0.991 |

In [71]:

sorted\_result**=**result**.**sort\_values(by**=**['Accuracy', 'f1\_score'],ascending**=False**)**.**reset\_index(drop**=True**)

In [72]:

sorted\_result

Out[72]:

|  | **ML Model** | **Accuracy** | **f1\_score** | **Recall** | **Precision** |
| --- | --- | --- | --- | --- | --- |
| **0** | Support Vector Machine | 0.957 | 0.963 | 0.982 | 0.966 |
| **1** | Decision Tree | 0.957 | 0.963 | 0.992 | 0.991 |
| **2** | Decision Tree | 0.957 | 0.963 | 0.992 | 0.991 |
| **3** | K-Nearest Neighbors | 0.944 | 0.950 | 0.962 | 0.996 |
| **4** | K-Nearest Neighbors | 0.944 | 0.950 | 0.962 | 0.996 |
| **5** | Logistic Regression | 0.924 | 0.933 | 0.947 | 0.927 |
| **6** | Naive Bayes Classifier | 0.583 | 0.420 | 0.291 | 0.996 |

In [73]:

**from** sklearn.ensemble **import** RandomForestClassifier

*# instantiate the model*

rfc**=** RandomForestClassifier(n\_estimators**=**10)

*# fit the model*

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

Out[73]:

RandomForestClassifier(n\_estimators=10)

In [74]:

**import** pickle

pickle**.**dump(rfc, open('model.pkl', 'wb'))

In [75]:

plt**.**figure(figsize**=**(9,7))

n\_features **=** X\_train**.**shape[1]

plt**.**barh(range(n\_features), gbc**.**feature\_importances\_, align**=**'center')

plt**.**yticks(np**.**arange(n\_features), X\_train**.**columns)

plt**.**title("Feature importances using permutation on full model")

plt**.**xlabel("Feature importance")

plt**.**ylabel("Feature")

plt**.**show()