Name:- Aniket Kumar

Sr.No:- 30

Roll no:- PF45

PRN No:- ~~Aniket~~ 1032171203

AML LAB ASSIGNMENT-3

Topic:- To study/implement Naive Bayes and Random Forest.

Algorithm used:- Random forest, Decision tree and NB.

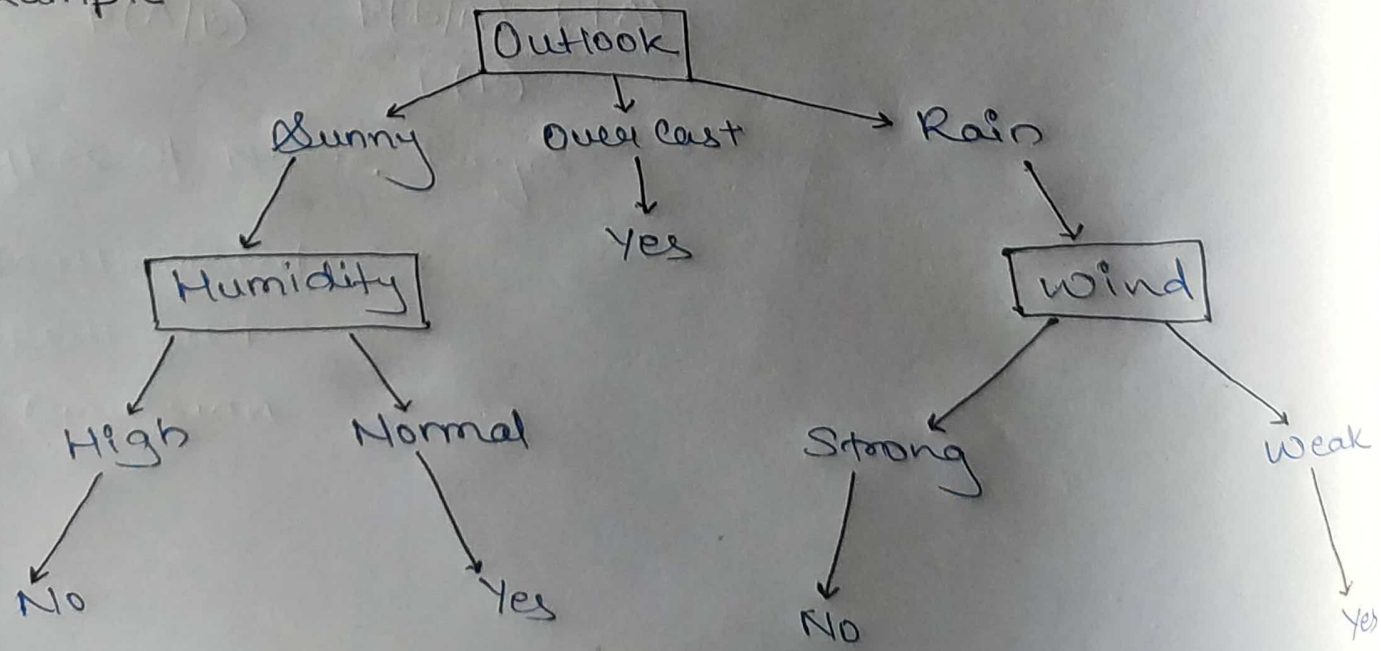
Theory:-

a) Decision Tree:-

It is a Supervised Machine learning algorithm, that can be used for both classification and regression

It is the most powerful and popular tool for classification and prediction.

Example



Pros:-

1) Generate understandable rules
2) Perform classification without much computation
3) Able to handle both continous and categorical variables.

Cons:

1) prone to errors in classification with many class and relatively small number of training examples.

b) Random Forest :-

It is a supervised learning algorithm which is used for both classification as well as regression. It is an esemble method which is better than single decision tree because it reduces overfiting by ap averaging the result.

Pros:

1) Work well for a large range of data item
2) Has less variance then decision tree.
3) very flexible and posses very high accuracy.

Cons:

1) Complex
2) Much harder and time consuming.
3) More computational resources required.

2) Feature Scaling:-

It is a technique to standardize the independent features present in the data in a fixed range.

3) Confusion Matrix:-

It is a better way to evaluate the performance of a classifier is to look at the Confusion matrix

Naive Bayes:-

Statistical method for classification. It is a Supervised learning method and can solve problems involving both categorical and continuous valued attribute

Principle:-

Probablistic machine learning model that is used for classification tasks.

Bayes Theorem:-

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

Types of Naive Bayes

1) Multinomial
2) Bernoulli
3) Gaussian

Advantages:-

- It is highly extensible algorithm which is very fast.
- It can be used for binary and multiclass classification
- It is famous algorithm for spam email classification.

Disadvantages:-

All the variables independent that contributes to the probability.

Application

1) Real time Prediction
2) Multi class classification
3) Text Classification

Conclusion:-

Different classifiers such as NB, RF, Regression, etc. where studied and their performance analysis was done.

NAME : Aniket Kumar

S.no : 30

PRN : 1032171203

ROLL : PF45

Subject : AML Lab Assignment 3

Problem Statement : Implement Naive Bayes Theorem and Random Forest Algorithm

Import the required python libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, KFold, Stratified
KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_sco
re, f1_score, log_loss
from sklearn.metrics import auc, roc_curve, roc_auc_score, precision_re
call_curve
from sklearn.metrics import fbeta_score, cohen_kappa_score

import warnings
warnings.filterwarnings("ignore")
SEED = 42
```

Import the data

In [2]:
```
column_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
df = pd.read_csv(r"C:\Users\Aniket\Desktop\Aml\3\pima-indians-diabetes.
csv", header = None, names = column_names)
```

Display info about the dataset

In [3]:
```
df.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
df.head()
```

Out[4]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnant    768 non-null int64
glucose     768 non-null int64
bp          768 non-null int64
skin        768 non-null int64
insulin     768 non-null int64
bmi         768 non-null float64
pedigree    768 non-null float64
age         768 non-null int64
label       768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
df.describe()
```

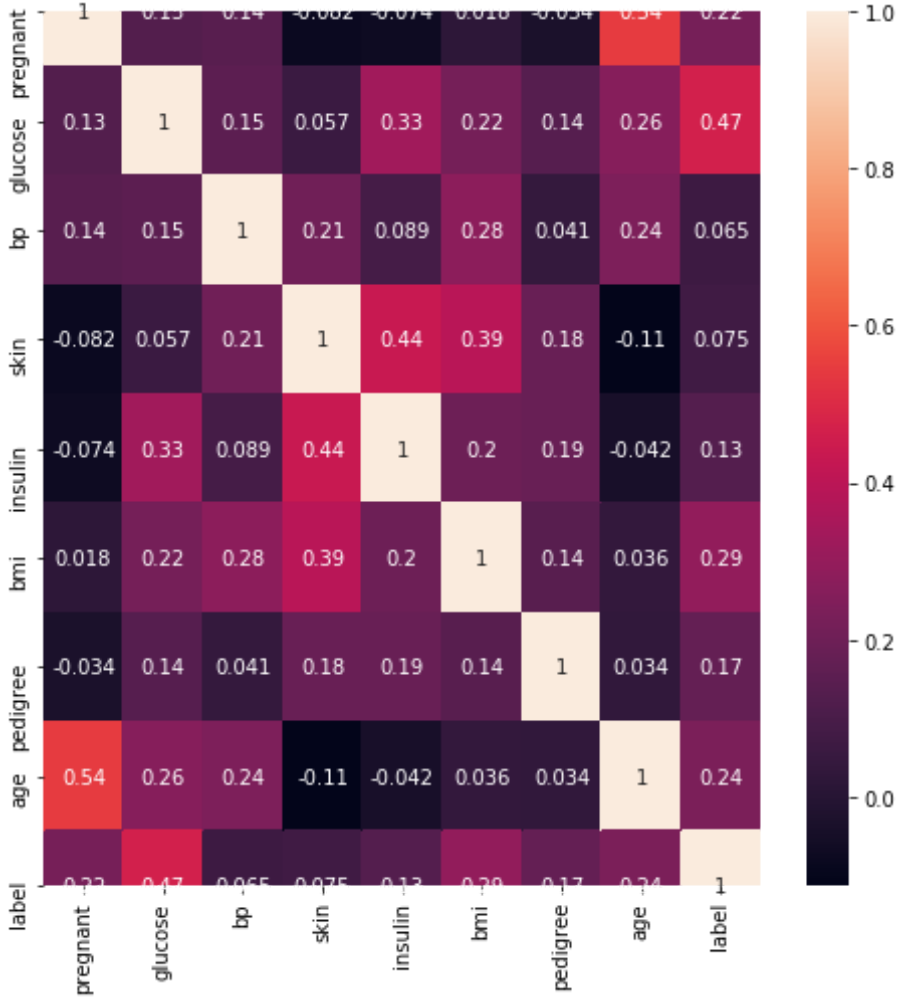
Out[6]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0

Correlation Matrix

In [7]:
```
fig = plt.figure(figsize = (8, 8))
correlationmatrix = df.corr()
sns.heatmap(correlationmatrix, annot = True)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2e1f2592cc8>
```



Data Preprocessing

In [8]:
```
df[['glucose', 'bp', 'skin', 'insulin', 'bmi']] = df[['glucose', 'bp',
'skin', 'insulin', 'bmi']].replace(0, np.NaN)
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
pregnant      0
glucose       5
bp           35
skin        227
insulin     374
bmi          11
pedigree      0
age           0
label         0
dtype: int64
```

In [10]:
```
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
df[['glucose', 'bp']] = imputer.fit_transform(df[['glucose', 'bp']])
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
df[['skin', 'insulin', 'bmi']] = imputer.fit_transform(df[['skin', 'ins
ulin', 'bmi']])
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
pregnant    0
glucose     0
bp          0
skin        0
insulin     0
bmi         0
pedigree    0
age         0
label       0
dtype: int64
```

Independant and Dependant variables

In [12]:
```
# Independant variables
feature_columns = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bm
i', 'pedigree', 'age']
X = df[feature_columns]
print("FEATURES: ", X.shape)
```

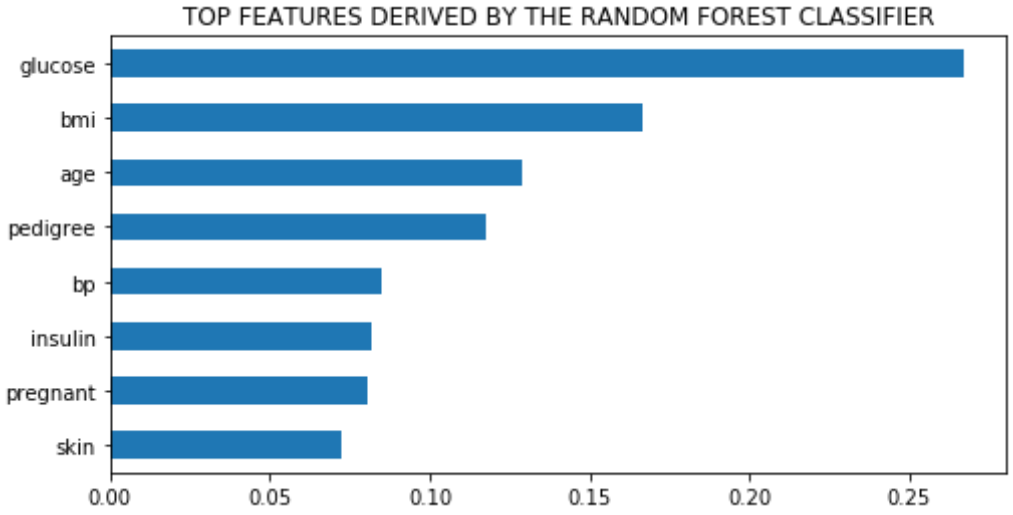
```
# Dependant variables
y = df.label
print("LABEL: ", y.shape)
```

```
FEATURES:  (768, 8)
LABEL:  (768,)
```

Feature Selection

In [13]:
```
rfc = RandomForestClassifier(random_state = SEED, n_estimators = 100)
model = rfc.fit(X, y)
(pd.Series(model.feature_importances_, index = X.columns)
   .nlargest(10)
   .plot(kind = 'barh', figsize = [8, 4])
   .invert_yaxis())
plt.title('TOP FEATURES DERIVED BY THE RANDOM FOREST CLASSIFIER')
```

Out[13]:
```
Text(0.5, 1.0, 'TOP FEATURES DERIVED BY THE RANDOM FOREST CLASSIFIER')
```



Feature Scaling

In [14]:
```
scaler = MinMaxScaler(feature_range = (0, 1))
standardized_X = pd.DataFrame(scaler.fit_transform(X), columns = featur
e_columns)
X = standardized_X
X.head()
```

Out[14]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age
0	0.352941	0.670968	0.489796	0.304348	0.133413	0.314928	0.234415	0.483333
1	0.058824	0.264516	0.428571	0.239130	0.133413	0.171779	0.116567	0.166667
2	0.470588	0.896774	0.408163	0.239130	0.133413	0.104294	0.253629	0.183333
3	0.058824	0.290323	0.428571	0.173913	0.096154	0.202454	0.038002	0.000000
4	0.000000	0.600000	0.163265	0.304348	0.185096	0.509202	0.943638	0.200000

In [15]:
```
X.describe()
```

Out[15]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	0.226180	0.501205	0.493930	0.240305	0.152250	0.291518	0.168179	0.2
std	0.198210	0.196361	0.123432	0.095557	0.103826	0.140597	0.141473	0.1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.058824	0.359677	0.408163	0.195652	0.129207	0.190184	0.070773	0.0
50%	0.176471	0.470968	0.491863	0.239130	0.133413	0.288344	0.125747	0.1
75%	0.352941	0.620968	0.571429	0.271739	0.136118	0.376278	0.234095	0.3
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

Split the data into Train and Test sets

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = SEED, stratify = y)
print('The shape of the of the resultant data sets are as follows...')
print('X_train: ', X_train.shape)
print('y_train: ', y_train.shape)
print('X_test : ', X_test.shape)
print('y_test : ', y_test.shape)
```

```
The shape of the of the resultant data sets are as follows...
X_train:  (614, 8)
y_train:  (614,)
X_test :  (154, 8)
y_test :  (154,)
```

Evaluate baseline model performance

In [17]:
```
kf = StratifiedKFold(n_splits = 5, shuffle = True, random_state = SEED)
def baseline_report(model, X_train, X_test, y_train, y_test, name):
    model.fit(X_train, y_train)
    accuracy  = np.mean(cross_val_score(model, X_train, y_train, cv = k
f, scoring = 'accuracy'))
    precision = np.mean(cross_val_score(model, X_train, y_train, cv = k
f, scoring = 'precision'))
    recall    = np.mean(cross_val_score(model, X_train, y_train, cv = k
f, scoring = 'recall'))
    f1score   = np.mean(cross_val_score(model, X_train, y_train, cv = k
f, scoring = 'f1'))
    rocauc   = np.mean(cross_val_score(model, X_train, y_train, cv = kf
, scoring = 'roc_auc'))

    y_pred    = model.predict(X_test)
    logloss   = log_loss(y_test, y_pred)

    if name != 'SVC' and name != 'LinearSVC':
        probs = model.predict_proba(X_test)
```

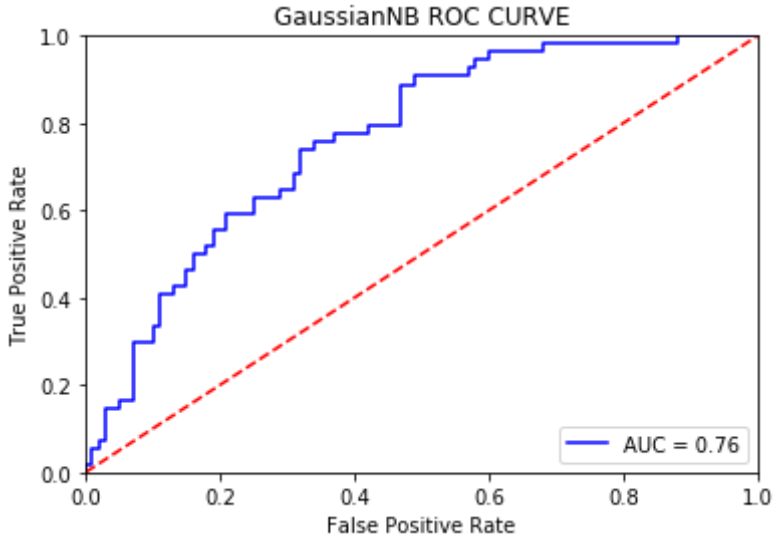
```
        preds = probs[:, 1]
        fpr, tpr, threshold = roc_curve(y_test, preds)
        roc_auc = auc(fpr, tpr)

        plt.title(name + ' ROC CURVE')
        plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
        plt.legend(loc = 'lower right')
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.show()

    df_model = pd.DataFrame({'model'    : [name],
                             'accuracy' : [accuracy],
                             'precision': [precision],
                             'recall'   : [recall],
                             'f1score'  : [f1score],
                             'rocauc'   : [rocauc],
                             'logloss'  : [logloss]})
    return df_model
```

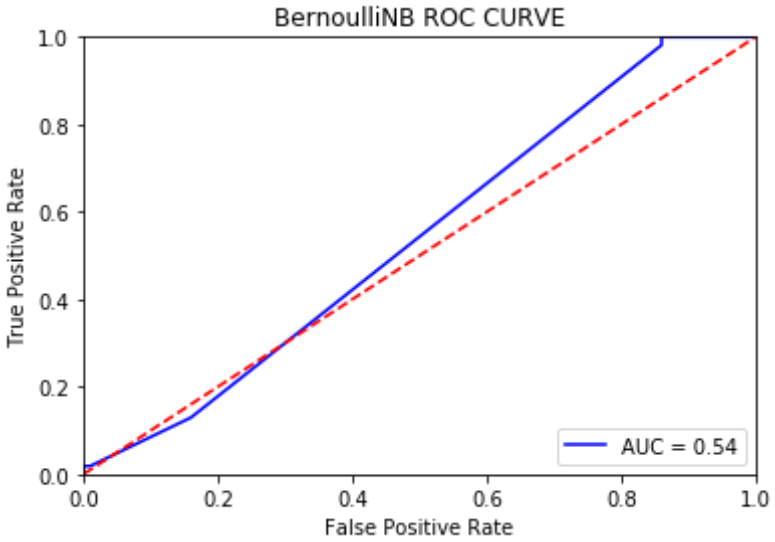
Initialize classifier objects

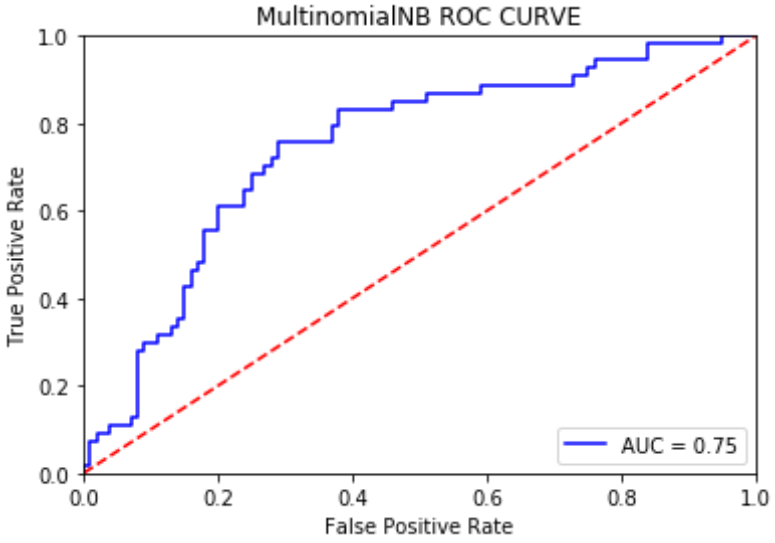
In [18]:
```
gnb = GaussianNB()
bnb = BernoulliNB()
mnb = MultinomialNB()
logit = LogisticRegression()
knn = KNeighborsClassifier()
decisiontree = DecisionTreeClassifier()
randomforest = RandomForestClassifier()
svc = SVC()
linearsvc = LinearSVC()
```

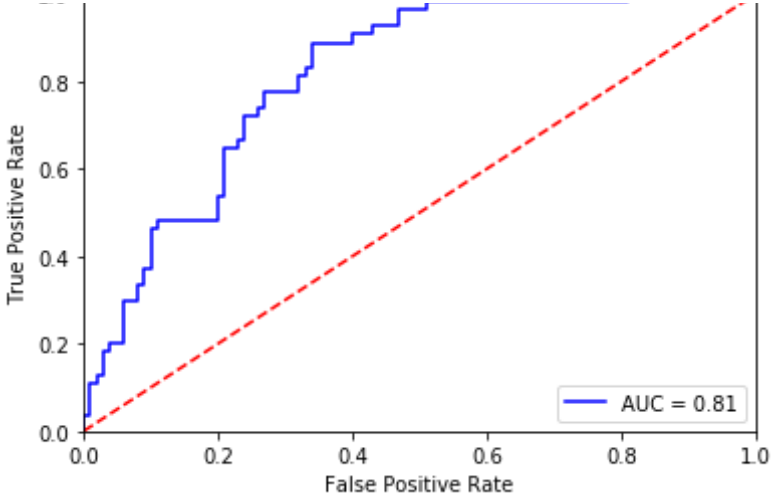
Analyze performance of classifiers

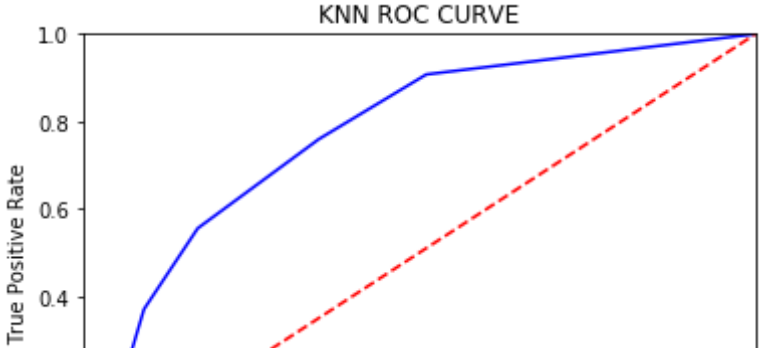
In [19]:
```
df_models = pd.concat([
    baseline_report(gnb, X_train, X_test, y_train, y_test, 'GaussianNB'
),
    baseline_report(bnb, X_train, X_test, y_train, y_test, 'BernoulliN
B'),
    baseline_report(mnb, X_train, X_test, y_train, y_test, 'Multinomial
NB'),
    baseline_report(logit, X_train, X_test, y_train, y_test, 'LogisticR
egression'),
    baseline_report(knn, X_train, X_test, y_train, y_test, 'KNN'),
    baseline_report(decisiontree, X_train, X_test, y_train, y_test, 'De
cisionTree'),
    baseline_report(randomforest, X_train, X_test, y_train, y_test, 'Ra
ndomForest'),
    baseline_report(svc, X_train, X_test, y_train, y_test, 'SVC'),
    baseline_report(linearsvc, X_train, X_test, y_train, y_test, 'Linea
rSVC')
], axis = 0).reset_index()

df_models = df_models.drop('index', axis = 1)
```

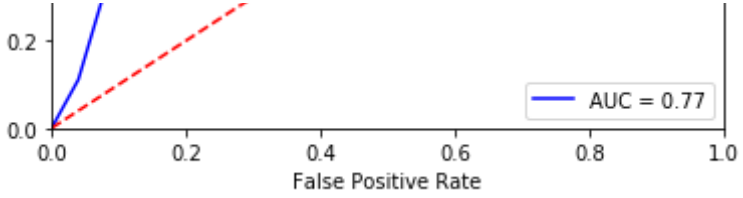


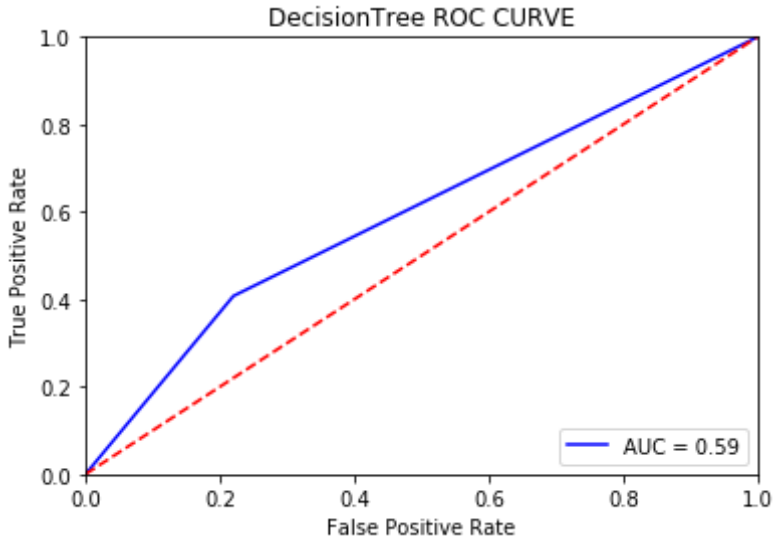


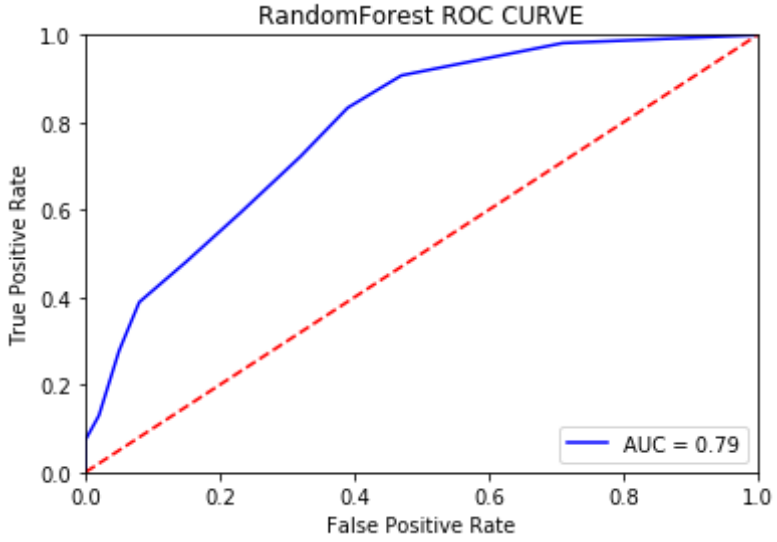












In [20]:
```
df_models
```

Out[20]:

	model	accuracy	precision	recall	f1score	rocauc	logloss
0	GaussianNB	0.752526	0.670439	0.579734	0.620708	0.827421	10.316912
1	BernoulliNB	0.649847	0.000000	0.000000	0.000000	0.543590	12.111005
2	MultinomialNB	0.651473	0.000000	0.000000	0.000000	0.700101	12.111000
3	LogisticRegression	0.775330	0.750004	0.533112	0.623074	0.843956	9.868305
4	KNN	0.765520	0.683805	0.616833	0.647705	0.806829	9.195477
5	DecisionTree	0.711769	0.587219	0.593909	0.587097	0.676347	12.111114
6	RandomForest	0.718286	0.691633	0.551717	0.604166	0.783288	9.644022
7	SVC	0.773677	0.755517	0.518937	0.615165	0.843992	8.971173
8	LinearSVC	0.778542	0.734191	0.570321	0.641551	0.843160	10.092588