In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import precision_score
from sklearn import metrics
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
import xgboost as xgb
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

```
/kaggle/input/data-antibody/data_antibody.csv
/kaggle/input/pcr-data/pcr_data.csv
/kaggle/input/both-covid-data/both_covid_data.csv
/kaggle/input/covid-data-gender/data_covid_fe.csv
/kaggle/input/covid-data-gender/both_covid_data_gender.csv
/kaggle/input/covid-data-gender/data_covid_ma.csv
```

Loading data_antibody!

In [2]:

```
data_antibody = pd.read_csv("/kaggle/input/data-antibody/data_antibody.csv")
data_antibody=data_antibody.astype(int)
data_antibody.head()
```

Out[2]:

	Symptom-Throat Pain	Symptom-Dyspnea	Symptom-Fever	Symptom-Cough	Symptom-Headache	Symptom-Taste Disorders	Symptom-Olfactory Disorders	Symptom-Coryza	Gender	Are you a health professional?	Class
0	0	1	0	0	1	0	1	0	0	1	0

	Symptom-Throat Pain	Symptom-Dyspnea	Symptom-Fever	Symptom-Cough	Symptom-Headache	Symptom-Taste Disorders	Symptom-Olfactory Disorders	Symptom-Coryza	Gender	Are you a health professional?	Class
1	1	0	0	0	0	1	1	1	1	0	0
2	0	0	1	0	0	0	1	0	0	1	0
3	0	1	0	0	1	1	0	1	1	0	0
4	1	1	0	0	1	1	1	0	0	1	0

Loading data_pcr!

In [3]:

```
data_pcr= pd.read_csv("/kaggle/input/pcr-data/pcr_data.csv")
data_pcr=data_pcr.astype(int)
data_pcr.head()
```

Out[3]:

	Symptom-Throat Pain	Symptom-Dyspnea	Symptom-Fever	Symptom-Cough	Symptom-Headache	Symptom-Taste Disorders	Symptom-Olfactory Disorders	Symptom-Coryza	Gender	Are you a health professional?	Class
0	0	1	0	0	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0	1	0	1	0
2	0	1	0	0	0	0	0	1	1	1	0
3	0	1	0	0	1	0	0	0	0	1	0
4	1	0	0	0	1	0	0	1	0	1	0

Loading both data!

In [4]:

```
data_both= pd.read_csv("/kaggle/input/both-covid-data/both_covid_data.csv")
#a.astype(float)
data_both=data_both.astype(int)
```

In [5]:

```
def RandomForest_classif(x_train,y_train):
    #Classification
    clf= RandomForestClassifier()
    clf=clf.fit(x_train, y_train)
    return clf

def Kneighbors_classif(x_train,y_train):
    #Classification
    clf= KNeighborsClassifier(n_neighbors=3)
    clf= clf.fit(x_train, y_train)
    return clf

def DecisionTree_classif(x_train,y_train):
    #Classification
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(x_train,y_train)
    return clf

def mpl_classif(x_train,y_train):
    clf =  MLPClassifier(max_iter=300,solver='lbfgs', alpha=1e-5, random_state=42)
    clf=clf.fit(x_train,y_train)
    return clf

def gb_classif(x_train,y_train):
    param_dist = {'n_estimators':500,'max_depth':5}
```

```
    clf=GradientBoostingClassifier(**param_dist)
    clf=clf.fit(x_train, y_train)
    return clf

def xgb_classif(x_train,y_train):
    param_dist = {'n_estimators':300,'max_depth':9,'min_child_weight': 2}

    clf = xgb.XGBClassifier(**param_dist)

    return clf.fit(x_train, y_train)

def svc_classif(x_train,y_train):
    regr = svm.SVC()
    regr=regr.fit(x_train, y_train)
    return regr
```

In [6]:

```
def calculate_metrics(x_train, x_test,y_train, y_test,data,k,clf):

        #prediction
        y_pred=clf.predict(x_test)

        #accuracy score
        acc=accuracy_score(y_test,y_pred)*100

        #confusion matrix
        tn, fp, fn, tp = confusion_matrix(y_test,y_pred).ravel()


        fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)

        #precision
        data.iloc[k:k+1,:1]=(tp/(tp+fp))*100

        data.iloc[k:k+1,1:2]=acc

        #recall
        data.iloc[k:k+1,2:3]=(tp/(tp+fn))*100


        #mean absolute error
        data.iloc[k:k+1,3:4]=mean_absolute_error(y_test, y_pred)*100

        #AUC
        data.iloc[k:k+1,4:5]= metrics.roc_auc_score(y_test,y_pred)*100


        return data
```

In [7]:

```
def calculate_prediction(x,y):

    dt=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    rf=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    dmlp=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    dgbm=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    xgboost=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    dt_svm=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))
    dt_knn=pd.DataFrame(columns=['P','ACC','R','MAE','AUC'],index=range(50))

    n=0
    for k in range(50):

        x_train, x_test, y_train, y_test = train_test_split(
        x, y, test_size=0.3,random_state=n,stratify=y)

        clf_svm=svc_classif(x_train,y_train)
        dt_svm=calculate_metrics(x_train, x_test,y_train, y_test, dt_svm,k,clf_svm)
```

```
        clf_knn=Kneighbors_classif(x_train,y_train)
        dt_knn=calculate_metrics(x_train, x_test,y_train, y_test, dt_knn,k,clf_knn)

        clf_dt=DecisionTree_classif(x_train,y_train)
        dt=calculate_metrics(x_train, x_test,y_train, y_test,dt,k,clf_dt)

        clf_rf=RandomForest_classif(x_train,y_train)
        rf=calculate_metrics(x_train, x_test,y_train, y_test,rf,k,clf_rf)

        clf_mlp=mpl_classif(x_train,y_train)
        dmlp=calculate_metrics(x_train, x_test,y_train, y_test,dmlp,k,clf_mlp)

        clf_gbm=gb_classif(x_train,y_train)
        dgbm=calculate_metrics(x_train, x_test,y_train, y_test,dgbm,k,clf_gbm)

        clf_xgboost=xgb_classif(x_train,y_train)
        xgboost=calculate_metrics(x_train, x_test,y_train, y_test,xgboost,k,clf_xgboost)

        n+=1


    return pd.DataFrame([dt.mean(),rf.mean(),dgbm.mean(),xgboost.mean(),dmlp.mean(),
                         dt_svm.mean(),dt_knn.mean()])
```

Test Antibody prediction!

In [8]:

```
d_ant=calculate_prediction(data_antibody.iloc[:,0:10],data_antibody['Class'])
d_ant=d_ant.rename(index={0:'Decision Tree',1:'Random Forest',2:'GBM', 3:'XGBoost',4:'Mlp',
                          5:'SVM',6:'KNN'})
d_ant=d_ant.astype(float)
d_ant
```

Out[8]:

	P	ACC	R	MAE	AUC
Decision Tree	98.858201	99.416370	100.0	0.583630	99.415957
Random Forest	99.074348	99.530249	100.0	0.469751	99.529990
GBM	99.074348	99.530249	100.0	0.469751	99.529990
XGBoost	98.866996	99.423488	100.0	0.576512	99.423151
Mlp	98.853282	99.416370	100.0	0.583630	99.416109
SVM	99.074348	99.530249	100.0	0.469751	99.529990
KNN	99.007279	99.494662	100.0	0.505338	99.494529

Test pcr prediction!

In [9]:

```
d_pcr=calculate_prediction(data_pcr.iloc[:,0:10],data_pcr['Class'])
d_pcr=d_pcr.rename(index={0:'Decision Tree',1:'Random Forest',2:'GBM', 3:'XGBoost',4:'Mlp',
                          5:'SVM',6:'KNN'})
d_pcr=d_pcr.astype(float)
d_pcr
```

Out[9]:

	P	ACC	R	MAE	AUC
Decision Tree	96.353979	97.080460	97.877395	2.919540	97.080460
Random Forest	96.978473	97.290805	97.846743	2.709195	97.290805

	P	ACC	R	MAE	AUC
Random Forest	96.978473	97.390805	97.840745	2.609195	97.390805
GBM	96.985010	97.402299	97.862069	2.597701	97.402299
XGBoost	96.706365	97.068966	97.478927	2.931034	97.068966
Mlp	97.684158	97.858238	98.053640	2.141762	97.858238
SVM	95.211460	94.275862	93.287356	5.724138	94.275862
KNN	94.759589	95.141762	95.586207	4.858238	95.141762

Both test prediction!

In [10]:

```
d=calculate_prediction(data_both.iloc[:,0:10],data_both['Class'])
d=d.rename(index={0:'Decision Tree',1:'Random Forest',2:'GBM', 3:'XGBoost',4:'Mlp',
                  5:'SVM',6:'KNN'})
d=d.astype(float)
d
```

Out[10]:

	P	ACC	R	MAE	AUC
Decision Tree	97.198459	97.628892	98.097617	2.371108	97.628758
Random Forest	98.504565	98.278954	98.052841	1.721046	98.279023
GBM	97.810649	98.009963	98.227106	1.990037	98.009950
XGBoost	98.192523	98.007472	97.823836	1.992528	98.007537
Mlp	98.080166	98.226650	98.386521	1.773350	98.226622
SVM	97.917837	96.077210	94.168249	3.922790	96.077586
KNN	96.796596	96.774595	96.763167	3.225405	96.774755