🤖 Supervised and Unsupervised Learning in Machine Learning

Machine Learning is the science of making computers learn and act like humans by feeding data and information without being explicitly programmed.

Machine learning algorithms are trained with training data. When new data comes in, they can make predictions and decisions accurately based on past data.

For example, whenever you ask Siri to do something, a powerful speech recognition converts the audio into its corresponding textual form. This is sent to the Apple servers for further processing where language processing algorithms are run to understand the user's intent. Then finally, Siri tells you the answer.

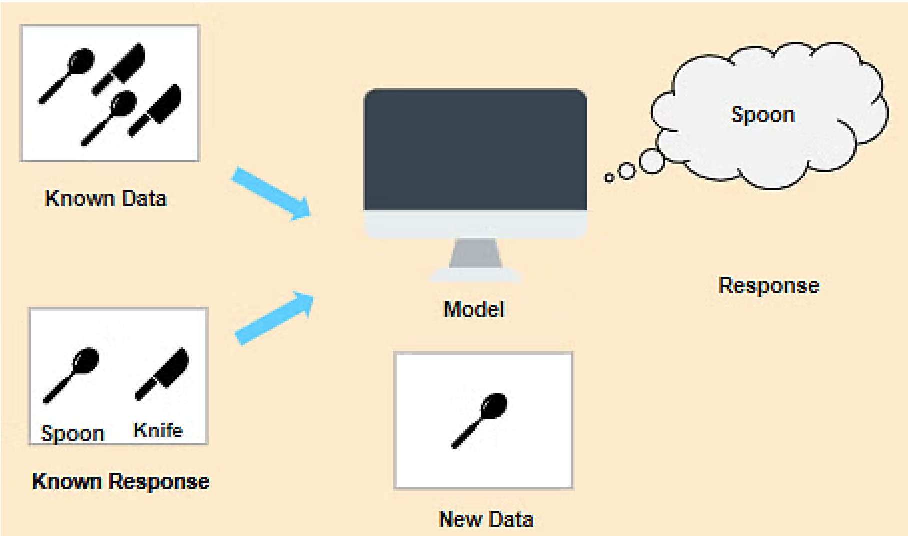
There are **two types** of machine learning:

1. Supervised Learning
2. Unsupervised Learning

Entrée [1]:
```
from IPython.display import Image
```

What is Supervised Learning?

In Supervised Learning, the machine learns under supervision. It contains a model that is able to predict with the help of a labeled dataset. A labeled dataset is one where you already know the target answer.



In this case, we have images that are labeled a spoon or a knife. This known data is fed to the machine, which analyzes and learns the association of these images based on its features such as shape, size, sharpness, etc. Now when a new image is fed to the machine without any label, the machine is able to predict accurately that it is a spoon with the help of the past data.

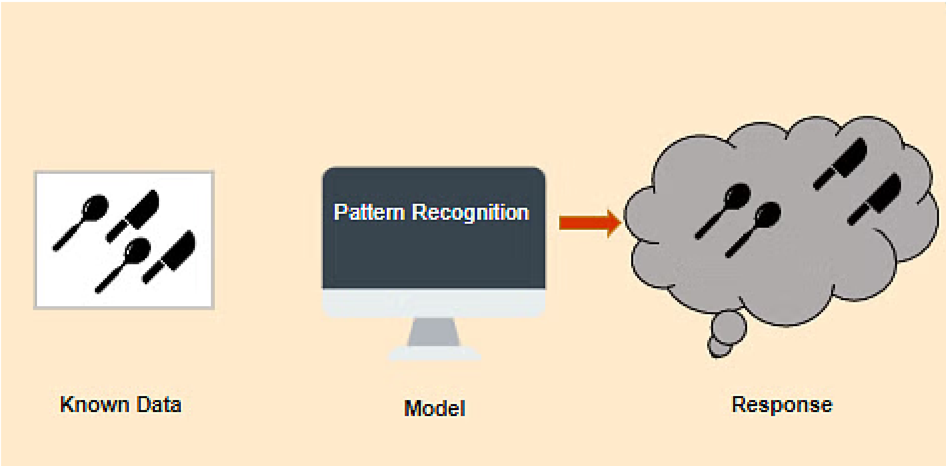
Supervised learning can be further divided into two types:

1. Classification

2. Regression

What is Unsupervised Learning?

In Unsupervised Learning, the machine uses unlabeled data and learns on itself without any supervision. The machine tries to find a pattern in the unlabeled data and gives a response.



Unsupervised learning can be further grouped into types:

1. Clustering
2. Association

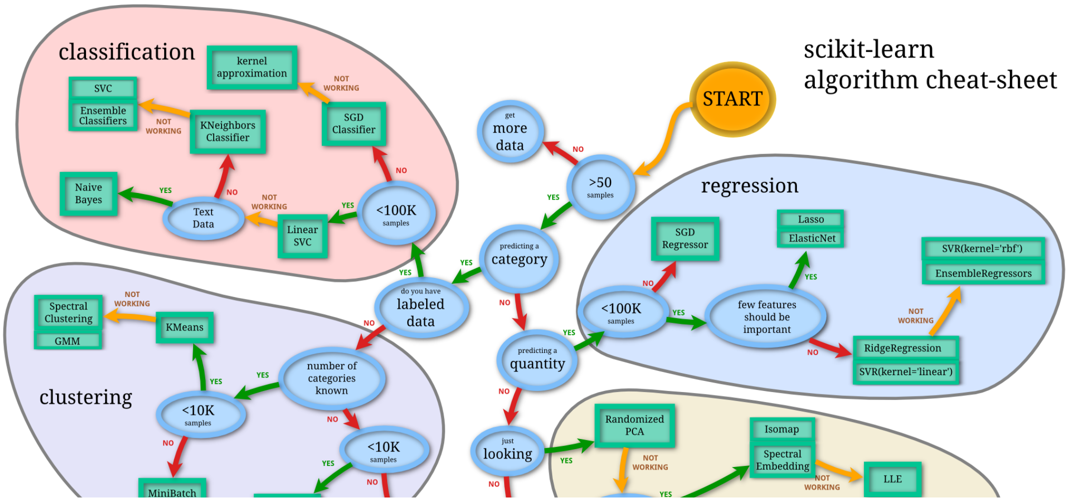
Difference Between Supervised and Unsupervised Learning

Supervised Learning	Unsupervised Learning
It uses known and labeled data as input	It uses unlabeled data as input
It has a feedback mechanism	It has no feedback mechanism
The most commonly used supervised learning algorithms are:	The most commonly used unsupervised learning algorithms are:
• Decision tree • Logistic regression • Support vector machine	• K-means clustering • Hierarchical clustering • Apriori algorithm

Choosing the right estimator

Often the hardest part of solving a machine learning problem can be finding the right estimator for the job.

Different estimators are better suited for different types of data and different problems.



In machine learning, Classification is used to split data into categories. But after cleaning and preprocessing the data and training our model, how do we know if our classification model performs well? That is where a confusion matrix comes into the picture.

A confusion matrix is used to measure the performance of a classifier in depth. In this simple guide to Confusion Matrix, we will get to understand and learn confusion matrices better.

✍🏻 MNIST dataset and performance measures

In this notebook, we would like to go through some of the ML algorithms. Precisely, we would like to evaluate and compare the following performance metrics :

- Confusion matrix
- Recall
- Precision
- FP Rate
- Specificity
- ROC curve

The measures will be taken on classification tasks on handwritten data.

The metrics will be implemented from scratch and will be compared to the metrics offered by the standard librairies *(scikit-learn)*.

🛑 KNN will be coded from scratch !

Importing librairies

Entrée [55]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.naive_bayes import GaussianNB,CategoricalNB
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
plt.style.use('seaborn')
from sklearn.svm import SVC
from sklearn import svm
import warnings
warnings.filterwarnings('ignore')
```

Lecture des fichiers de données

Pour ce TP, nous allons lire les données à partir d'un fichier csv.

Entrée [7]:

```
Data = pd.read_csv('Data/breast.csv')
Data
```

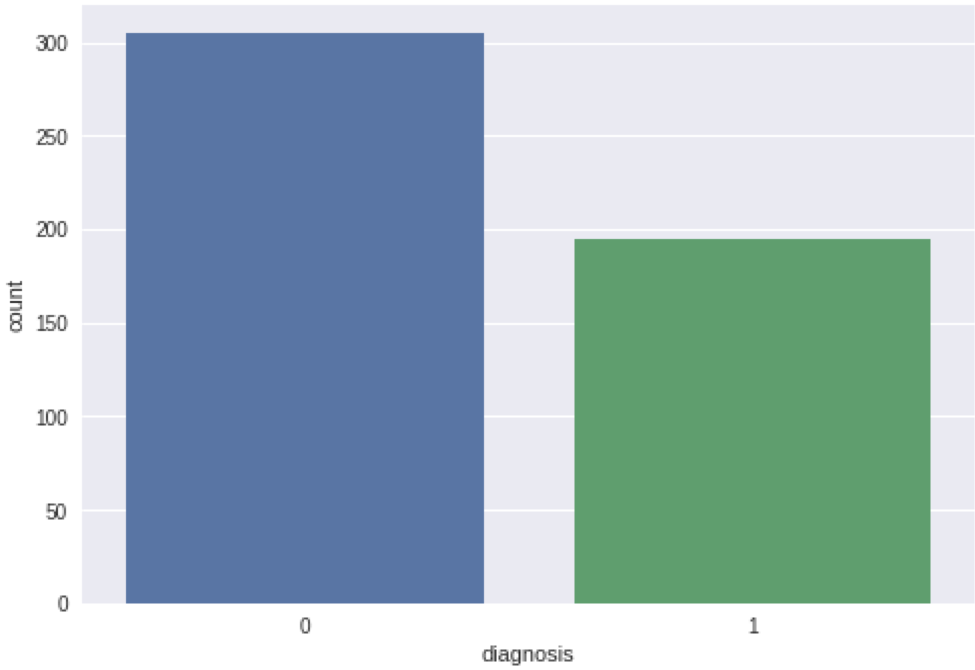
Out[7]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compacti
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	
495	14.87	20.21	96.12	680.9	0.09587	
496	12.65	18.17	82.69	485.6	0.10760	
497	12.47	17.31	80.45	480.1	0.08928	
498	18.49	17.52	121.30	1068.0	0.10120	
499	20.59	21.24	137.80	1320.0	0.10850	

500 rows × 19 columns

Entrée [10]:
```
sns.countplot(x='diagnosis',data=Data)
```

Out[10]: <AxesSubplot:xlabel='diagnosis', ylabel='count'>



Entrée [11]:
```
X = Data.drop("diagnosis", axis = 1)
Y = Data['diagnosis']
```

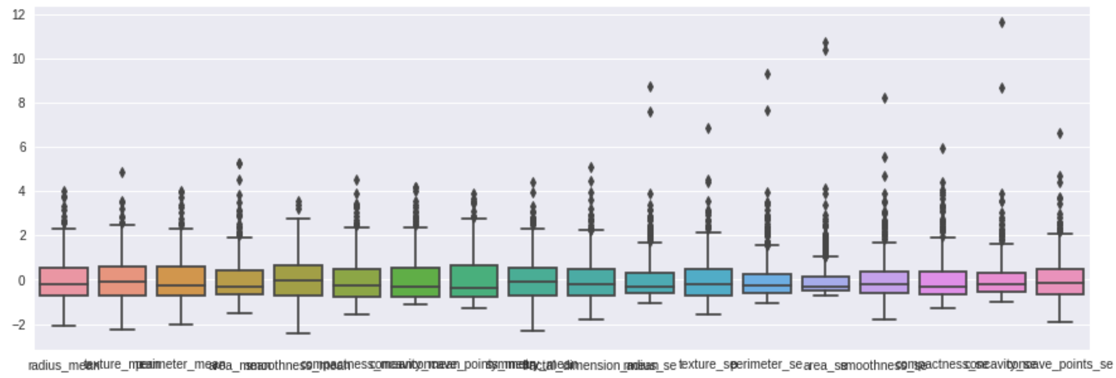
Normalisation

Entrée [12]:
```
Newdata2 = X.copy()
for column in X.columns:
    if Newdata2[column].dtype == 'float64' or Newdata2[column].
        Newdata2[column] = (X[column] - X[column].mean()) / X[c
        M=[]
l=Newdata2.shape[0]
for i in range(l):
    c=0
    for column in Newdata2.columns:
        if Newdata2[column].dtype == 'float64':
            c=c+1
            M.append(Newdata2[column].iloc[i])
newar=np.asarray(M)
newdat=newar.reshape(l,c)
X = pd.DataFrame(newdat,columns = [column for column in Newdata

```

Box-Plot

Entrée [13]:
```
plt.figure(figsize=(15,5))
sns.boxplot(data=X)
```

Out[13]: <AxesSubplot:>



📊📂The training set (data & labels)

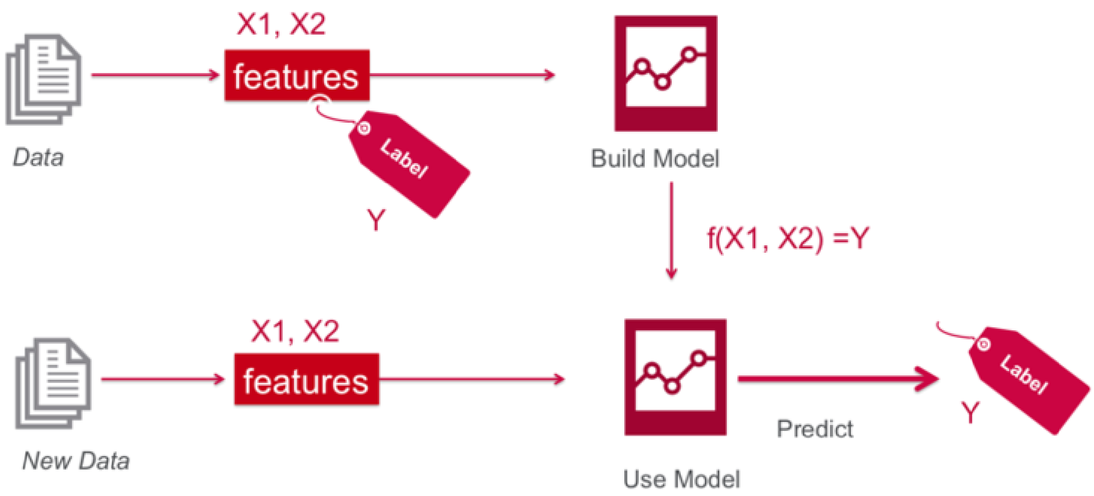
📜 Data

Entrée [14]:
```
x_train, x_test, y_train ,y_test = train_test_split(X,Y,test_si
```

🔖 Labels

In machine learning, data labeling is the process of identifying raw data (images, text files, videos, etc.) and adding one or more meaningful and informative labels to provide context so that a machine learning model can learn from it.

In our case, we have a csv file comprising of labels (from 1 to 10).

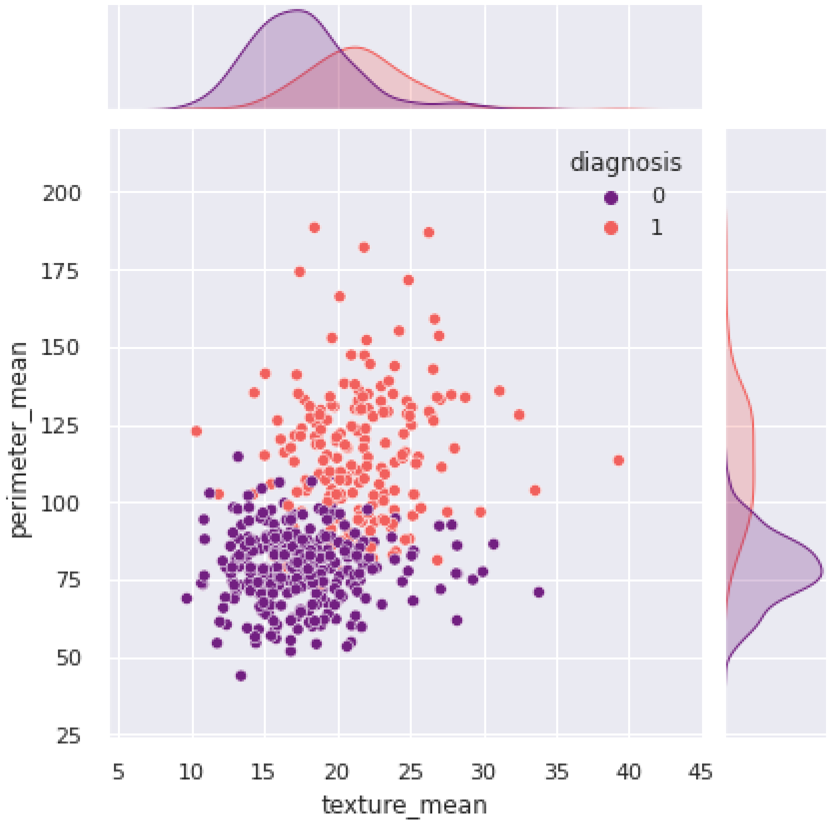


Let's dive into our `labels.csv` file and see how our labels look like :

Entrée [17]:

```
#plt.figure(figsize=(20,10))
sns.set_theme()
sns.jointplot(data=Data,x='texture_mean',y='perimeter_mean',hue
```

Out[17]: <seaborn.axisgrid.JointGrid at 0x7f5fc1796c70>



By convention, **label 10** corresponds to **0**. This is used in order to facilitate heavy computations.

📉 Performance measures

In this part, we're going to define the general performance measures all along with **their implementation from scratch**.

What Are Confusion Matrices, and Why Do We Need Them?

Classification Models have multiple categorical outputs. Most error measures will calculate the total error in our model, but we cannot find individual instances of errors in our model. The model might misclassify some categories more than others, but we cannot see this using a standard accuracy measure.

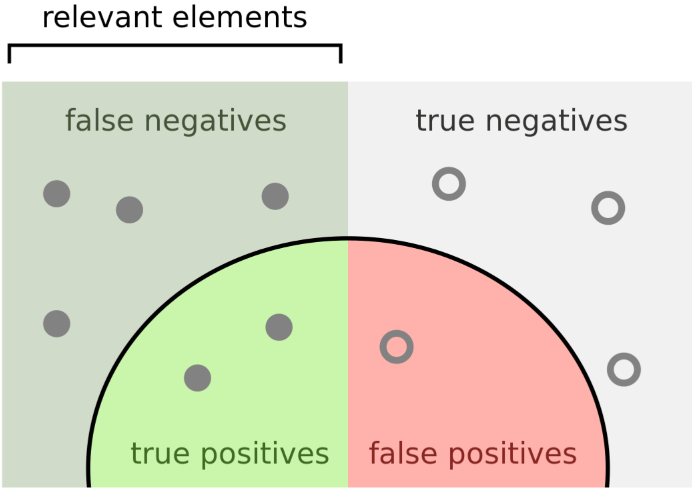
Furthermore, suppose there is a significant class imbalance in the given data. In that case, i.e., a class has more instances of data than the other classes, a model might predict the majority class for all cases and have a high accuracy score; when it is not predicting the minority classes. This is where confusion matrices are useful.

A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes.

It plots a table of all the predicted and actual values of a classifier.

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive **+**	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

False Positives (FP-Type 1 error) vs False Negatives (FN-Type 2 error)



⚫ Confusion Matrix

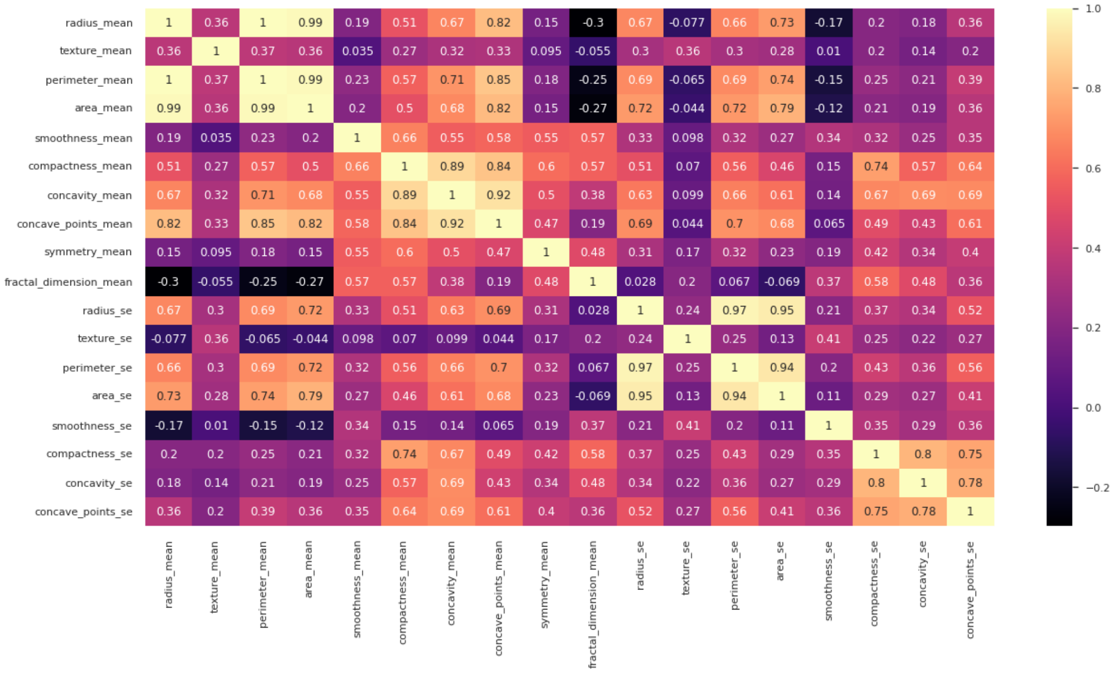
In our case, we have 8 classes, so the matrix is 8*8.

The matrix will look like this one :

Entrée [18]:

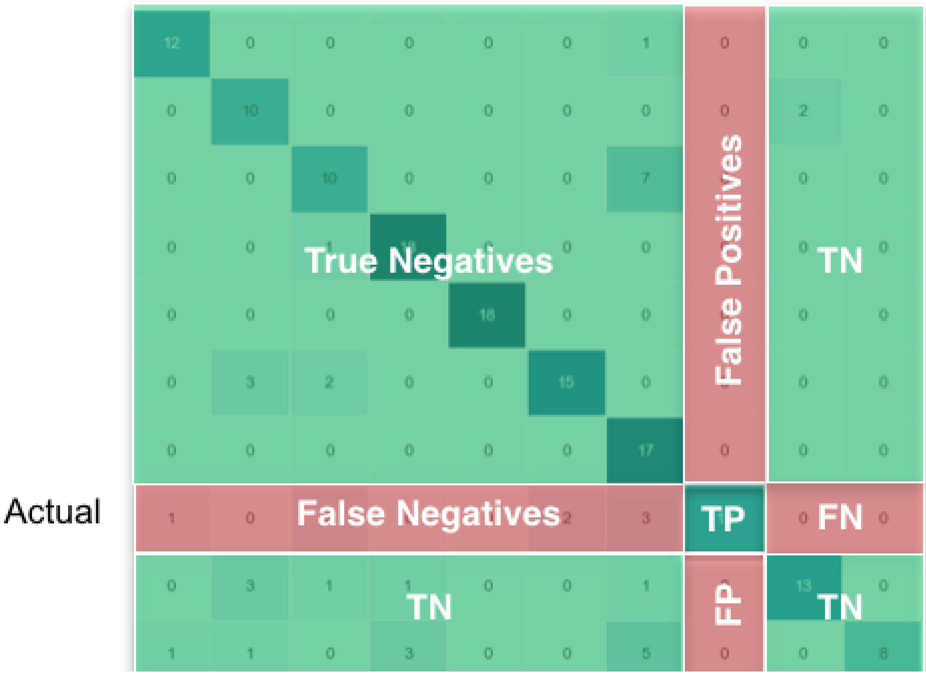
```
plt.figure(figsize=(20,10))
corr = X.corr() #utilisé pour trouver la corrélation par paires
sns.heatmap(corr, annot=True, cmap="magma")
```

Out[18]: <AxesSubplot:>



Now, we got our confusion matrix.

In order to compute the main performance measure, we need to extract from that matrix the FP,FN, TP and TN.



⚫ Accuracy

How many values did we predict correctly? How many true predictions out of all samples there are?

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All Samples}}$$

📒 Classification

Méthode 1 : KNN

Ici il faudra implémenter la méthode, puis la tester et vérifier les métriques en variant le nombre K

Entrée [19]:

```
KNN=KNeighborsClassifier(n_neighbors=3)
KNN.fit(x_train,y_train)
```

Out[19]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Entrée [20]:

```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train SCORE", "Test SCORE"])
z=str(int(KNN.score(x_train,y_train)*100))+"%"
v=str(int(KNN.score(x_test,y_test)*100))+"%"
x.add_row(["KNN",z,v])
print(x)
```

```
+-------+-------------+------------+
| Model | Train SCORE | Test SCORE |
+-------+-------------+------------+
|  KNN  |     96%     |    90%     |
+-------+-------------+------------+
```

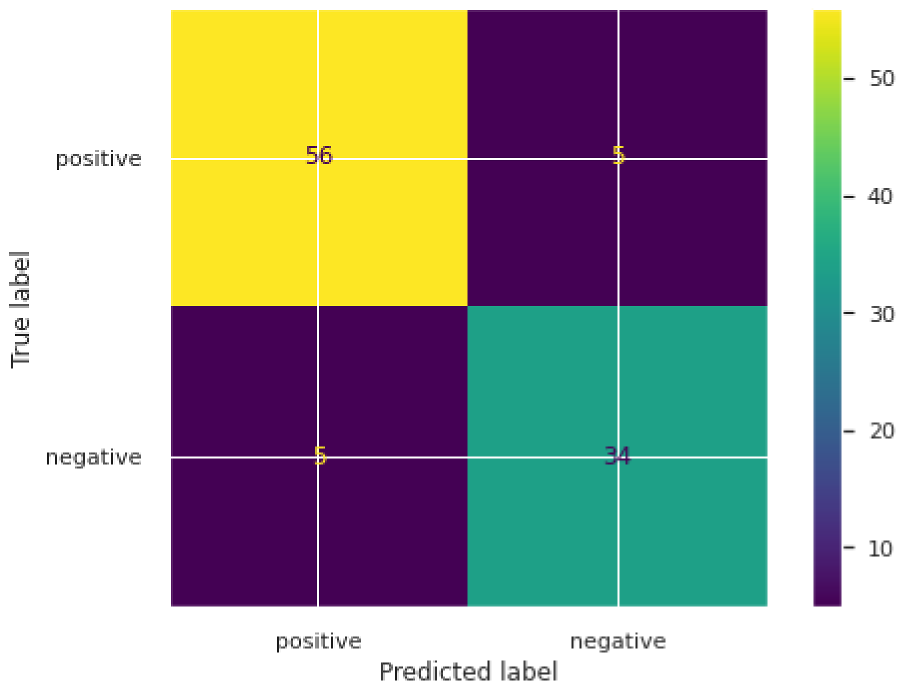
Confusion Matrices KNN

Entrée [21]:

```
y_pred=KNN.predict(x_test)
print(metrics.classification_report(y_test,y_pred))
display=metrics.ConfusionMatrixDisplay((metrics.confusion_matri
display.plot()
```

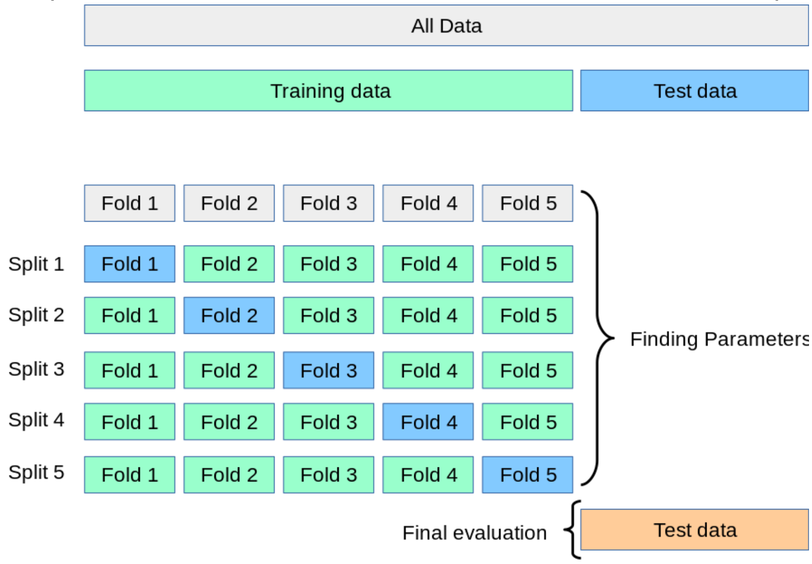
```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        61
           1       0.87      0.87      0.87        39

    accuracy                           0.90       100
   macro avg       0.89      0.89      0.89       100
weighted avg       0.90      0.90      0.90       100
```

Out[21]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f5fc1737f70>



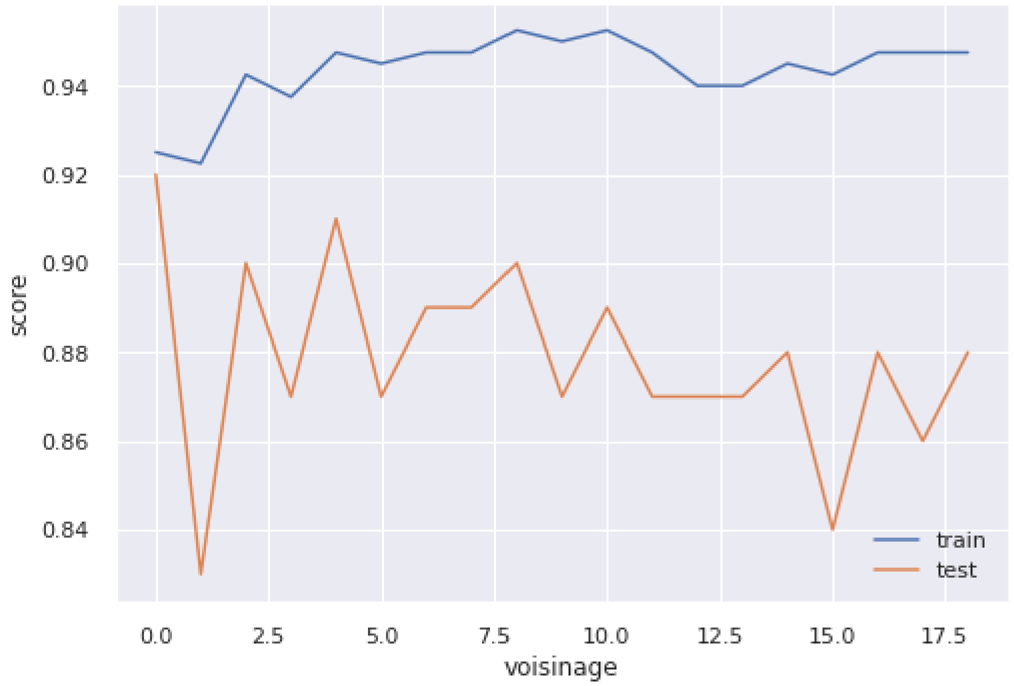
Cross-validation: evaluating estimator performance

La mesure de performance rapportée par la validation croisée k -fold est alors la moyenne des valeurs calculées dans la boucle. Cette approche peut être coûteuse en termes de calcul, mais ne gaspille pas trop de données (comme c'est le cas lors de la correction d'un ensemble de validation arbitraire), ce qui constitue un avantage majeur dans des problèmes tels que l'inférence inverse où le nombre d'échantillons est très petit.



Entrée [22]:
```
val_scortrain=[]
val_scortest=[]
for k in range(1,20):
    score=cross_val_score(KNeighborsClassifier(k),x_train,y_tra
    val_scortrain.append(score)
    score=cross_val_score(KNeighborsClassifier(k),x_test,y_test
    val_scortest.append(score)
plt.plot(val_scortrain,label='train')
plt.plot(val_scortest,label='test')
plt.ylabel('score')
plt.xlabel('voisinage')
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x7f5fc037ae50>

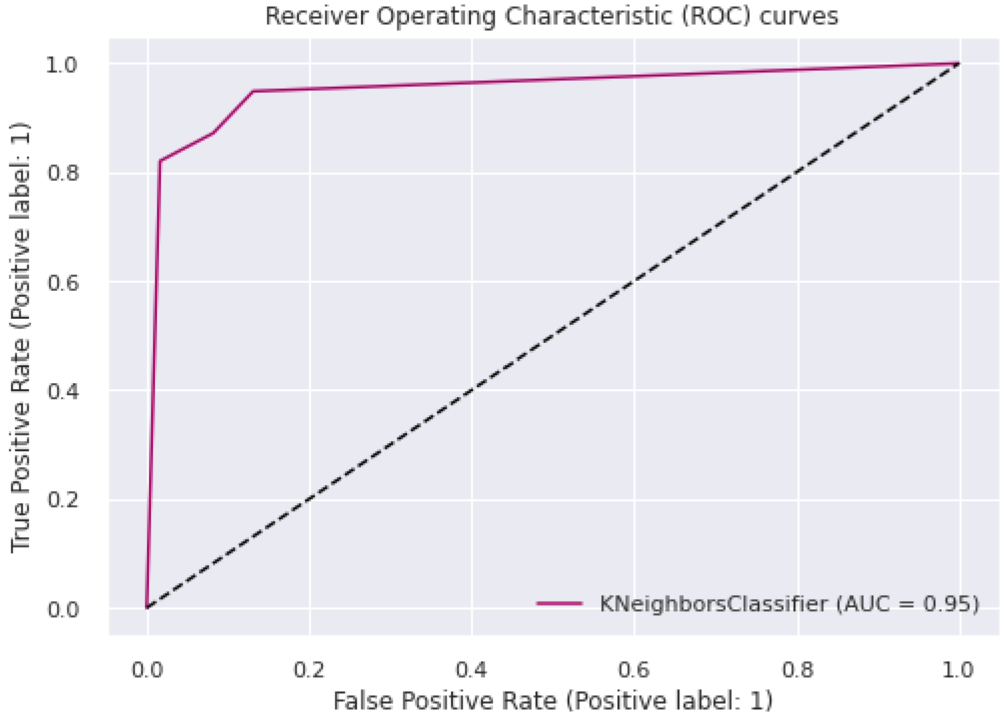


la courbe (ROC AUC)

Il s'agit d'une fonction générale, étant donné des points sur une courbe. Pour calculer l'aire sous la courbe ROC

Entrée [23]:

```
# Make predictions on the test set
y_pred = KNN.predict(x_test)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
y_test_binarized = label_binarize(y_test, classes=np.unique(y_t
if len(y_test_binarized[0])<=1:
    fig,ax_roc = plt.subplots()
    RocCurveDisplay.from_estimator(KNN, x_test, y_test, ax=ax_r
    plt.plot([0,1],[0,1],'--',c='black')
    ax_roc.set_title("Receiver Operating Characteristic (ROC) c
    #ax_roc.grid(linestyle="--",c='black')
    plt.legend()
    plt.show()
elif (len(y_test_binarized[0])>1):
    C=["#4B2991","#952EA0","#D44292","#F66D7A","#F6A97A"]
    Classses=np.unique(y_test)
    pred_prob=SVM.predict_proba(X_test)
    fpr = {}
    tpr = {}
    thresh ={}
    roc_auc = dict()
    n_classe = len(Classses)
    for i in range (n_classe):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[
        roc_auc[i] = auc (fpr[i], tpr[i])
    # traçage
        plt.plot(fpr[i], tpr[i],label='%s vs Rest (AUC=%0.2f)'%
    plt.plot([0,1], [0,1],linestyle='--', c='black')
    plt.xlim([0,1])
    plt.ylim([0,1.05])
    plt.title('Receiver Operting Characteristic(ROC) courves')
    plt.xlabel('False Rate')
    plt.ylabel('True Rate')
    plt.legend()
    plt.show()
```



Méthode 2:Naive Bayes

Entrée [24]:

```
Naive=GaussianNB()
Naive.fit(x_train,y_train)
```

Out[24]:

```
▾ GaussianNB

GaussianNB()
```

Entrée [25]:

```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train SCORE", "Test SCORE"])
z=str(int(Naive.score(x_train,y_train)*100))+"%"
v=str(int(Naive.score(x_test,y_test)*100))+"%"
x.add_row(["Naive Bayes",z,v])
print(x)
```

```
+-------------+-------------+------------+
|    Model    | Train SCORE | Test SCORE |
+-------------+-------------+------------+
| Naive Bayes |     91%     |    88%     |
+-------------+-------------+------------+
```

Confusion Matrices Naive Bayes

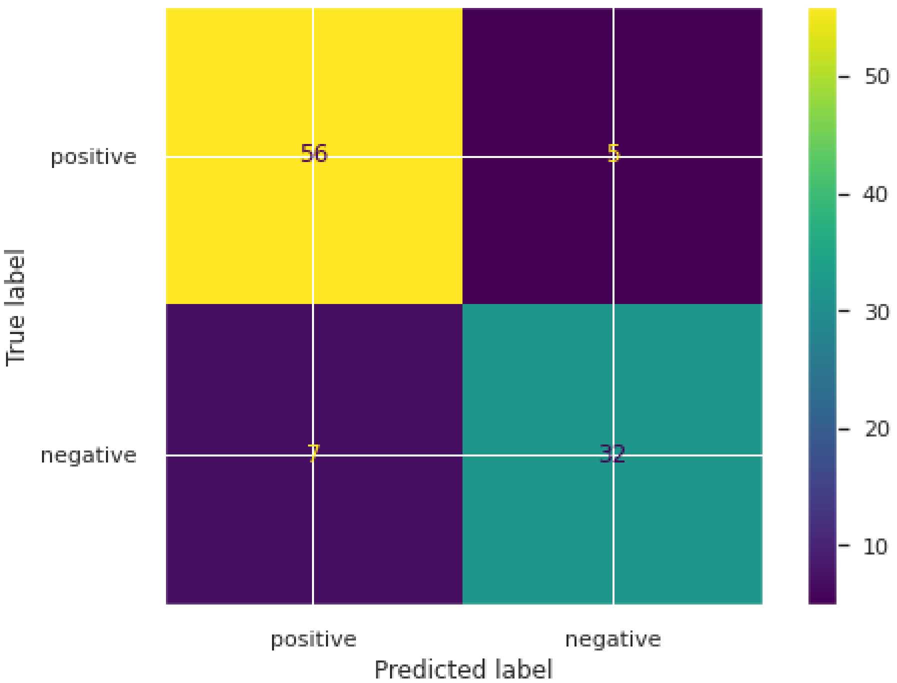
Entrée [26]:

```
y_pred=Naive.predict(x_test)
print(metrics.classification_report(y_test,y_pred))
display=metrics.ConfusionMatrixDisplay((metrics.confusion_matri
display.plot()
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90        61
           1       0.86      0.82      0.84        39

    accuracy                           0.88       100
   macro avg       0.88      0.87      0.87       100
weighted avg       0.88      0.88      0.88       100
```

Out[26]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5fc0385af0>
```



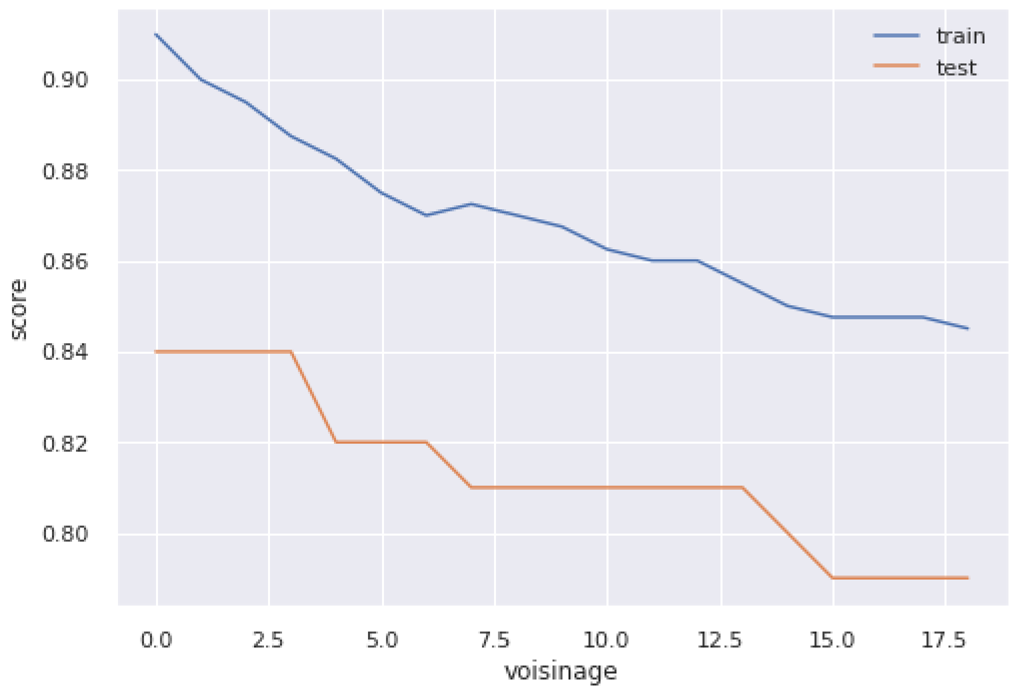
Cross-validation: evaluating estimator performance

Entrée [27]:

```
val_scortrain=[]
val_scortest=[]
for k in range(1,20):
    m=float(k/4)
    score=cross_val_score(GaussianNB(var_smoothing=m),x_train,y
    val_scortrain.append(score)
    score=cross_val_score(GaussianNB(var_smoothing=m),x_test,y_
    val_scortest.append(score)
plt.plot(val_scortrain,label='train')
plt.plot(val_scortest,label='test')
plt.ylabel('score')
plt.xlabel('voisinage')
plt.legend()
```

Out[27]:

```
<matplotlib.legend.Legend at 0x7f5fc01f5b50>
```

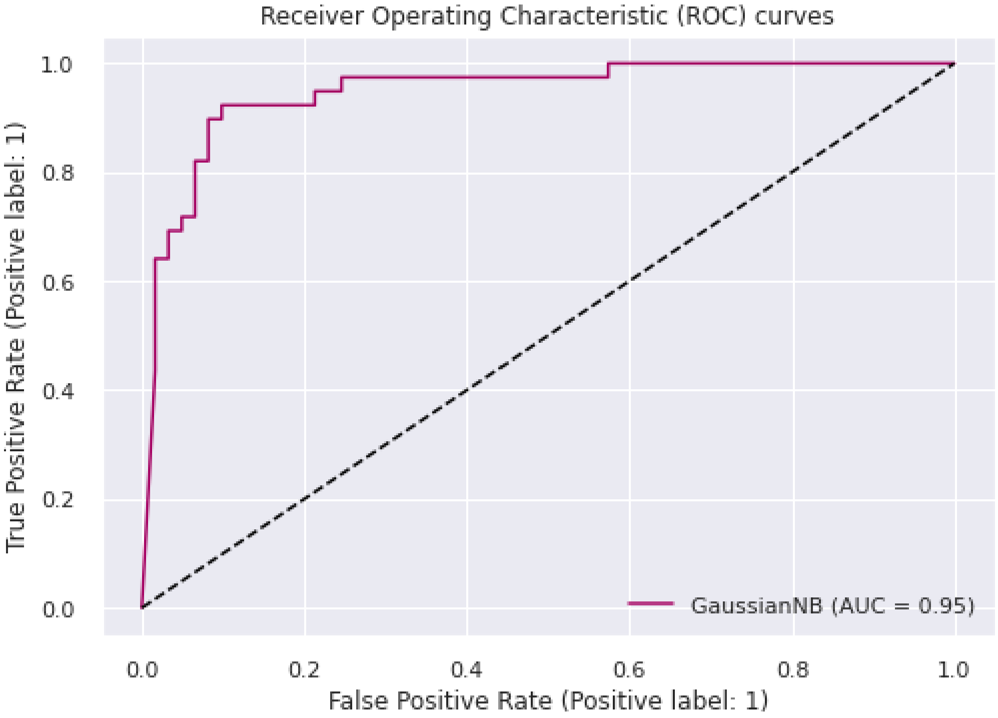


la courbe (ROC AUC)

Il s'agit d'une fonction générale, étant donné des points sur une courbe. Pour calculer l'aire sous la courbe ROC

Entrée [28]:

```
# Make predictions on the test set
y_pred = Naive.predict(x_test)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
y_test_binarized = label_binarize(y_test, classes=np.unique(y_t
if len(y_test_binarized[0])<=1:
    fig,ax_roc = plt.subplots()
    RocCurveDisplay.from_estimator(Naive, x_test, y_test, ax=ax
    plt.plot([0,1],[0,1],'--',c='black')
    ax_roc.set_title("Receiver Operating Characteristic (ROC) c
    #ax_roc.grid(linestyle="--",c='black')
    plt.legend()
    plt.show()
elif (len(y_test_binarized[0])>1):
    C=["#4B2991","#952EA0","#D44292","#F66D7A","#F6A97A"]
    Classses=np.unique(y_test)
    pred_prob=Naive.predict_proba(X_test)
    fpr = {}
    tpr = {}
    thresh ={}
    roc_auc = dict()
    n_classe = len(Classses)
    for i in range (n_classe):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[
        roc_auc[i] = auc (fpr[i], tpr[i])
    # traçage
        plt.plot(fpr[i], tpr[i],label='%s vs Rest (AUC=%0.2f)'%
    plt.plot([0,1], [0,1],linestyle='--', c='black')
    plt.xlim([0,1])
    plt.ylim([0,1.05])
    plt.title('Receiver Operting Characteristic(ROC) courves')
    plt.xlabel('False Rate')
    plt.ylabel('True Rate')
    plt.legend()
    plt.show()
```



Méthode 3:DecisionTreeClassifier

Entrée [37]:

```
#entropy ,gini
AD=DecisionTreeClassifier(criterion = 'entropy',max_depth=10)
AD.fit(x_train,y_train)
```

Out[37]:

```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=10)
```

Entrée [39]:

```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train SCORE", "Test SCORE"])
z=str(int(AD.score(x_train,y_train)*100))+"%"
v=str(int(AD.score(x_test,y_test)*100))+"%"
x.add_row([" DecisionTree",z,v])
print(x)
```

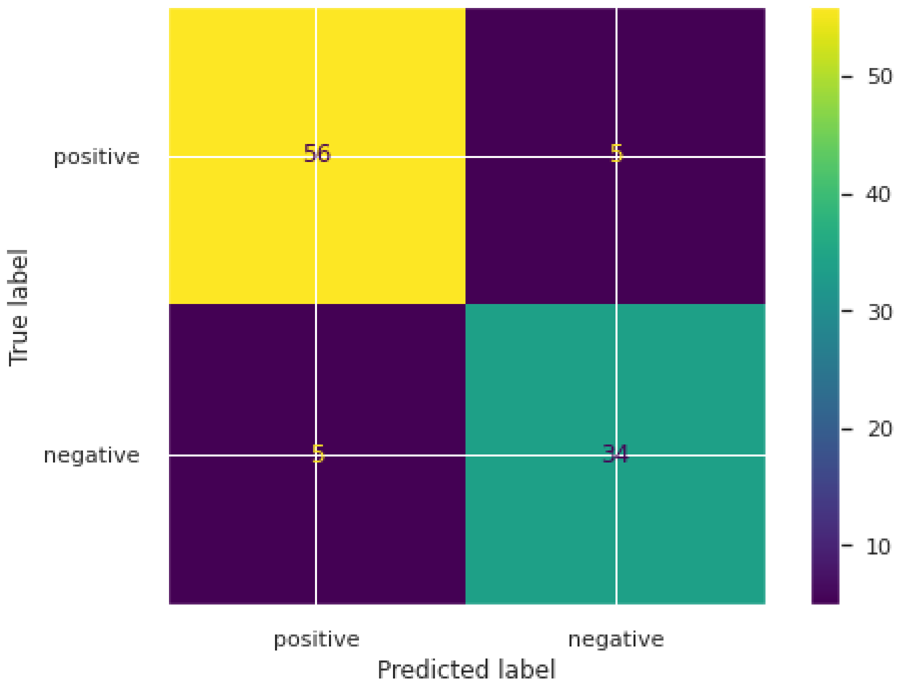
```
+--------------+------------+------------+
|    Model     | Train SCORE | Test SCORE |
+--------------+------------+------------+
|  DecisionTree |    100%    |    90%     |
+--------------+------------+------------+
```

Confusion Matrices DecisionTreeClassifier

Entrée [40]:

```
y_pred =AD.predict(x_test)
print(metrics.classification_report(y_test,y_pred))
display=metrics.ConfusionMatrixDisplay((metrics.confusion_matri
display.plot()
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        61
           1       0.87      0.87      0.87        39

    accuracy                           0.90       100
   macro avg       0.89      0.89      0.89       100
weighted avg       0.90      0.90      0.90       100
```

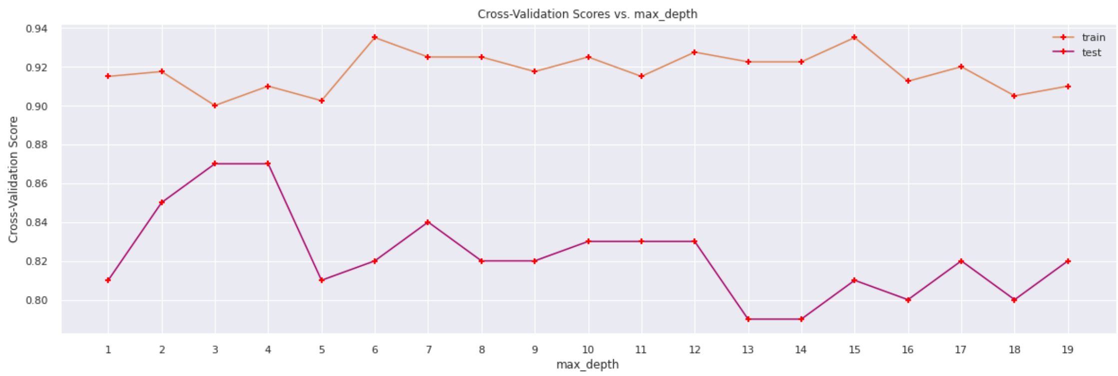


Cross-validation: evaluating estimator performance

Entrée [45]:

```
max_depth_values = range(1, 20)
cross_val_scores = []
cross_val_scorestest=[]
for depth in max_depth_values:
    clf = DecisionTreeClassifier(criterion = 'entropy',max_dept
    scores_train = cross_val_score(clf, x_train, y_train, cv=5)
    cross_val_scores.append(np.mean(scores_train))
    scores_test = cross_val_score(clf, x_test, y_test, cv=5)
    cross_val_scorestest.append(np.mean(scores_test))

plt.figure(figsize=(20, 6))
plt.plot(max_depth_values, cross_val_scores, marker='P',c="C1",
plt.plot(max_depth_values, cross_val_scorestest, marker='P',c='
plt.title('Cross-Validation Scores vs. max_depth')
plt.xlabel('max_depth')
plt.ylabel('Cross-Validation Score')
plt.xticks(max_depth_values)
plt.grid(True)
plt.legend()
plt.show
```

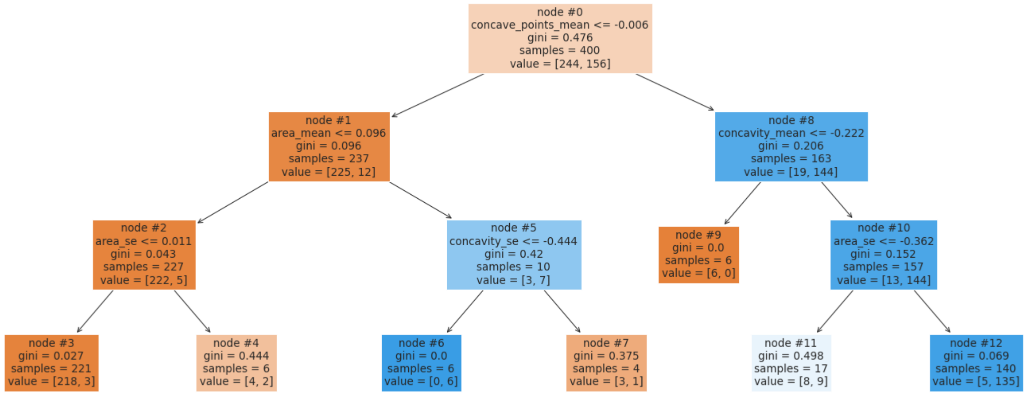
Out[45]: `<function matplotlib.pyplot.show(close=None, block=None)>`



DecisionTreeClassifier

Entrée [43]:

```
AD=DecisionTreeClassifier(max_depth = 3)
AD.fit(x_train,y_train)
plt.figure(figsize=(25,10))
plot_tree(AD,feature_names = x_test.columns,filled = True,node_
plt.show()
```

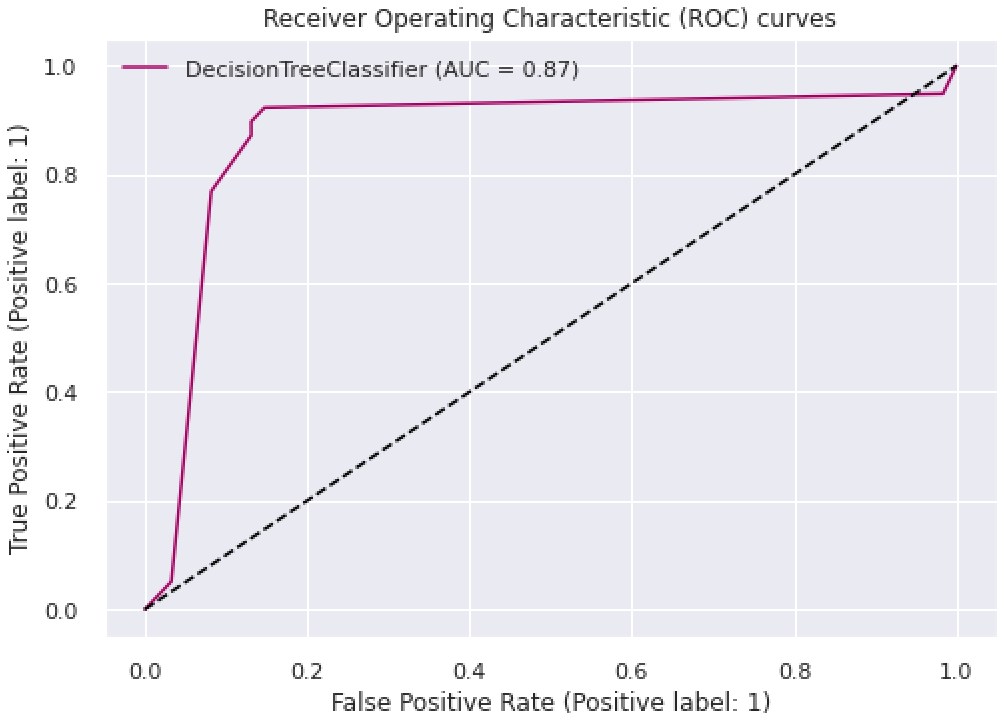


la courbe (ROC AUC)

Il s'agit d'une fonction générale, étant donné des points sur une courbe. Pour calculer l'aire sous la courbe ROC

Entrée [46]:

```python
# Make predictions on the test set
y_pred = AD.predict(x_test)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
y_test_binarized = label_binarize(y_test, classes=np.unique(y_t
if len(y_test_binarized[0])<=1:
    fig,ax_roc = plt.subplots()
    RocCurveDisplay.from_estimator(AD, x_test, y_test, ax=ax_ro
    plt.plot([0,1],[0,1],'--',c='black')
    ax_roc.set_title("Receiver Operating Characteristic (ROC) c
    #ax_roc.grid(linestyle="--",c='black')
    plt.legend()
    plt.show()
elif (len(y_test_binarized[0])>1):
    C=["#4B2991","#952EA0","#D44292","#F66D7A","#F6A97A"]
    Classses=np.unique(y_test)
    pred_prob=AD.predict_proba(X_test)
    fpr = {}
    tpr = {}
    thresh ={}
    roc_auc = dict()
    n_classe = len(Classses)
    for i in range (n_classe):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[
        roc_auc[i] = auc (fpr[i], tpr[i])
    # traçage
        plt.plot(fpr[i], tpr[i],label='%s vs Rest (AUC=%0.2f)'%
    plt.plot([0,1], [0,1],linestyle='--', c='black')
    plt.xlim([0,1])
    plt.ylim([0,1.05])
    plt.title('Receiver Operting Characteristic(ROC) courves')
    plt.xlabel('False Rate')
    plt.ylabel('True Rate')
    plt.legend()
    plt.show()
```



Méthode 4:Reseau Neurones

Entrée [47]:

```
# Create an instance of MLPClassifier
mlf = MLPClassifier(hidden_layer_sizes=(45), max_iter=1000)
# Train the model
mlf.fit(x_train, y_train)
```

Out[47]:

```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=45, max_iter=1000)
```

Entrée [48]:

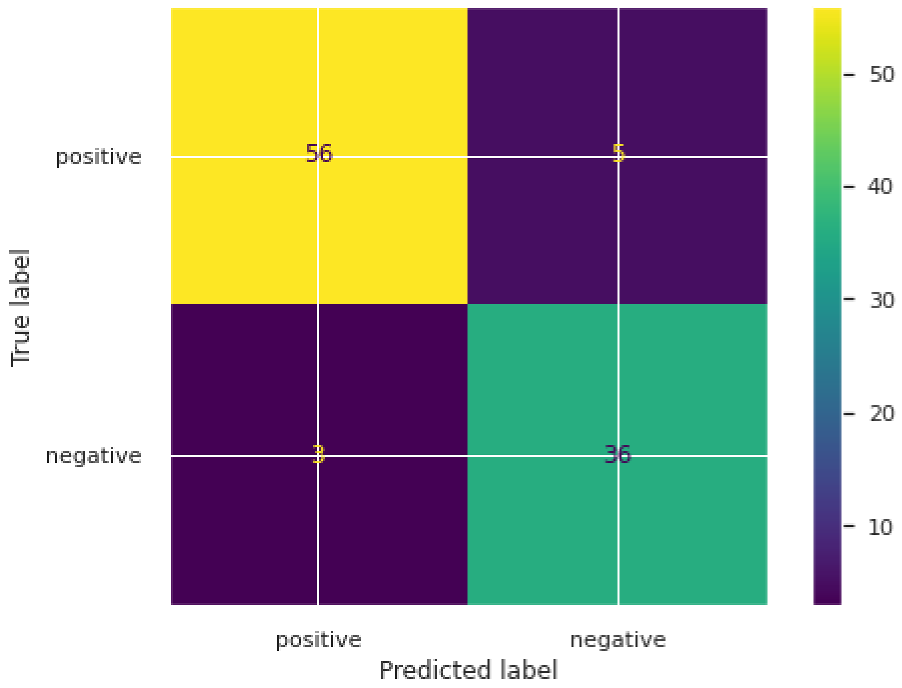
```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train SCORE", "Test SCORE"])
z=str(int(mlf.score(x_train,y_train)*100))+"%"
v=str(int(mlf.score(x_test,y_test)*100))+"%"
x.add_row([" Reseau Neurones",z,v])
print(x)
```

```
+------------------+-------------+------------+
|      Model       | Train SCORE | Test SCORE |
+------------------+-------------+------------+
|  Reseau Neurones |     99%     |    92%     |
+------------------+-------------+------------+
```

Confusion Matrices Reseau Neurones

Entrée [49]:
```
y_pred =mlf.predict(x_test)
print(metrics.classification_report(y_test,y_pred))
display=metrics.ConfusionMatrixDisplay((metrics.confusion_matri
display.plot()
plt.show()

```

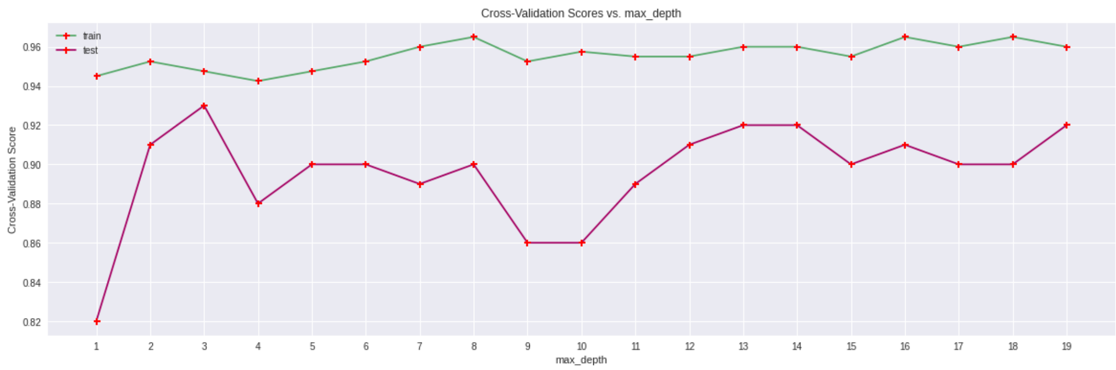
```
              precision    recall  f1-score   support

           0       0.95      0.92      0.93        61
           1       0.88      0.92      0.90        39

    accuracy                           0.92       100
   macro avg       0.91      0.92      0.92       100
weighted avg       0.92      0.92      0.92       100
```



Cross-validation: evaluating estimator performance

Entrée [59]:
```
max_depth_values = range(1, 20)
cross_val_scores = []
cross_val_scorestest=[]
for depth in max_depth_values:
    clf = mlf = MLPClassifier(hidden_layer_sizes=(depth), max_i
    scores_train = cross_val_score(clf, x_train, y_train, cv=5)
    cross_val_scores.append(np.mean(scores_train))
    scores_test = cross_val_score(clf, x_test, y_test, cv=5)
    cross_val_scorestest.append(np.mean(scores_test))

plt.figure(figsize=(20, 6))
plt.plot(max_depth_values, cross_val_scores, marker='P',c="C1",
plt.plot(max_depth_values, cross_val_scorestest, marker='P',c='

plt.title('Cross-Validation Scores vs. max_depth')
plt.xlabel('max_depth')
plt.ylabel('Cross-Validation Score')
plt.xticks(max_depth_values)
plt.grid(True)
plt.legend()
plt.show
```

Out[59]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

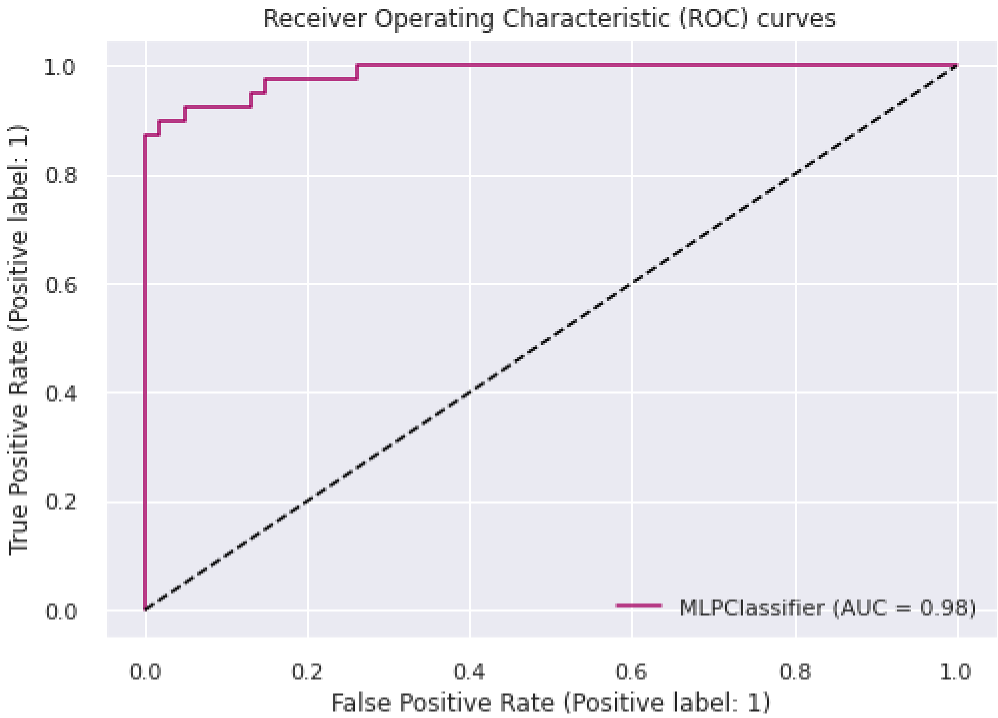


la courbe (ROC AUC)

Il s'agit d'une fonction générale, étant donné des points sur une courbe. Pour calculer l'aire sous la courbe ROC

Entrée [50]:

```
# Make predictions on the test set
y_pred = mlf.predict(x_test)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
y_test_binarized = label_binarize(y_test, classes=np.unique(y_t
if len(y_test_binarized[0])<=1:
    fig,ax_roc = plt.subplots()
    RocCurveDisplay.from_estimator(mlf, x_test, y_test, ax=ax_r
    plt.plot([0,1],[0,1],'--',c='black')
    ax_roc.set_title("Receiver Operating Characteristic (ROC) c
    #ax_roc.grid(linestyle="--",c='black')
    plt.legend()
    plt.show()
elif (len(y_test_binarized[0])>1):
    C=["#4B2991","#952EA0","#D44292","#F66D7A","#F6A97A"]
    Classses=np.unique(y_test)
    pred_prob=mlf.predict_proba(X_test)
    fpr = {}
    tpr = {}
    thresh ={}
    roc_auc = dict()
    n_classe = len(Classses)
    for i in range (n_classe):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[
        roc_auc[i] = auc (fpr[i], tpr[i])
    # traçage
        plt.plot(fpr[i], tpr[i],label='%s vs Rest (AUC=%0.2f)'%
    plt.plot([0,1], [0,1],linestyle='--', c='black')
    plt.xlim([0,1])
    plt.ylim([0,1.05])
    plt.title('Receiver Operting Characteristic(ROC) courves')
    plt.xlabel('False Rate')
    plt.ylabel('True Rate')
    plt.legend()
    plt.show()
```



Méthode 5:SVM

Entrée [51]:

```
# Create an instance of MLPClassifier

svm = svm.SVC(kernel='rbf',C=0.1)
# Train the model
svm.fit(x_train, y_train)
```

Out[51]:

```
▾ SVC
SVC(C=0.1)
```

Entrée [52]:

```
from prettytable import PrettyTable

x = PrettyTable(["Model", "Train SCORE", "Test SCORE"])
z=str(int(svm.score(x_train,y_train)*100))+"%"
v=str(int(svm.score(x_test,y_test)*100))+"%"
x.add_row([" SVM",z,v])
print(x)
```

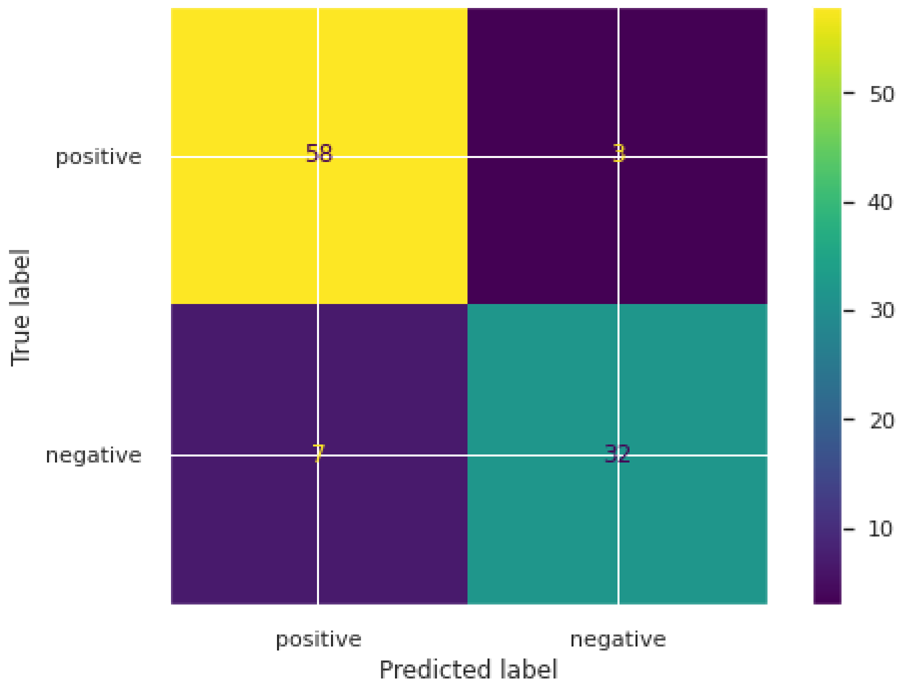
```
+-------+------------+------------+
| Model | Train SCORE | Test SCORE |
+-------+------------+------------+
|   SVM |     94%     |    90%     |
+-------+------------+------------+
```

Confusion Matrices SVM

Entrée [53]:

```
y_pred =svm.predict(x_test)
print(metrics.classification_report(y_test,y_pred))
display=metrics.ConfusionMatrixDisplay((metrics.confusion_matri
display.plot()
plt.show()
```

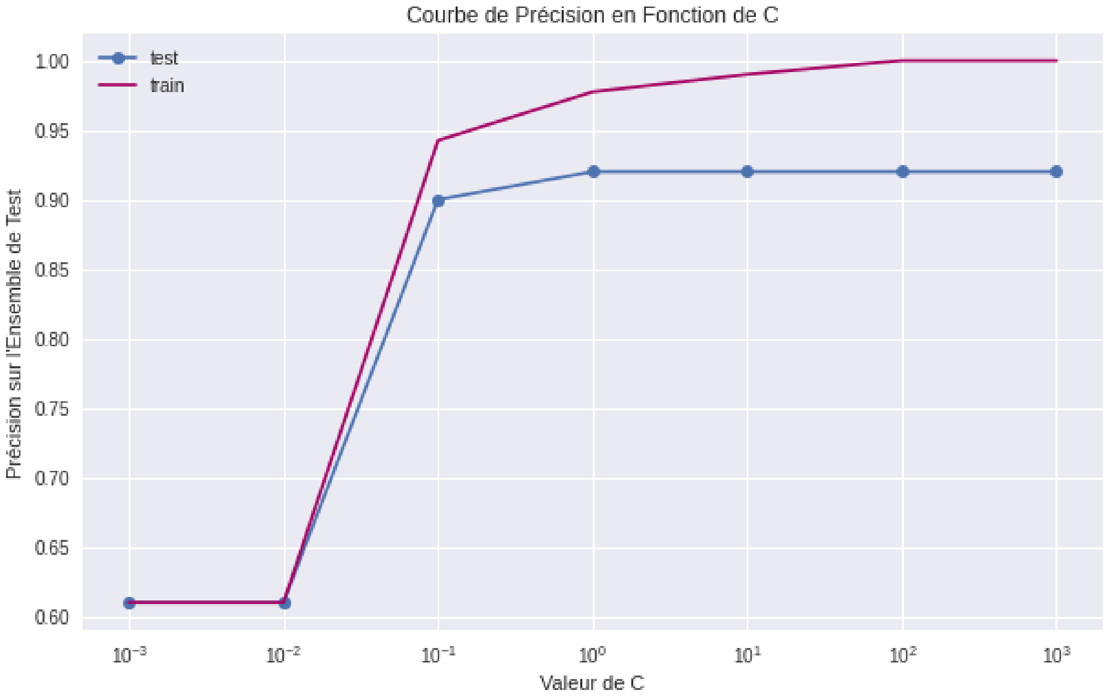
```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92        61
           1       0.91      0.82      0.86        39

    accuracy                           0.90       100
   macro avg       0.90      0.89      0.89       100
weighted avg       0.90      0.90      0.90       100
```



Cross-validation: evaluating estimator performance

Entrée [56]:

```
 #Définir les valeurs de C que vous souhaitez tester
from sklearn.metrics import accuracy_score
C_values = np.logspace(-3, 3, 7)

# Entraîner le modèle pour différentes valeurs de C et enregist
accuracies = []
train=[]
for C in C_values:
    # Créer un classificateur SVM avec noyau linéaire et la val
    svm_model = svm.SVC(kernel='rbf', C=C)

    # Entraîner le modèle sur l'ensemble d'entraînement
    svm_model.fit(x_train, y_train)

    # Faire des prédictions sur l'ensemble de test
    y_pred = svm_model.predict(x_test)
    y_predt = svm_model.predict(x_train)

    # Calculer la précision et l'enregistrer
    accuracy = accuracy_score(y_test, y_pred)
    accuracyt = accuracy_score(y_train, y_predt)
    accuracies.append(accuracy)
    train.append(accuracyt)

# Tracer la courbe de précision en fonction de C
plt.figure(figsize=(10, 6))
plt.semilogx(C_values, accuracies, marker='o',label='test')
plt.semilogx(C_values, train, c='#A50062',label='train',marker=
plt.title('Courbe de Précision en Fonction de C')
plt.xlabel('Valeur de C')
plt.ylabel('Précision sur l\'Ensemble de Test')
plt.grid(True)
plt.legend()
plt.show()

# Trouver la meilleure valeur de C
best_C = C_values[np.argmax(accuracies)]
print(f"Meilleur paramètre C : {best_C}")
```

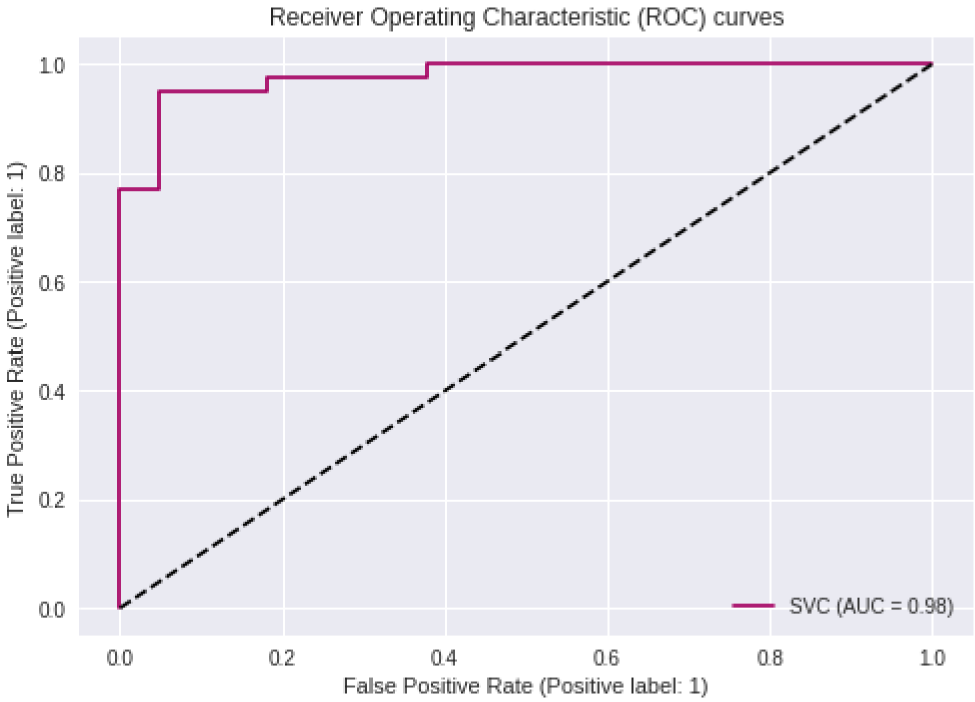



```
Meilleur paramètre C : 1.0
```

la courbe (ROC AUC)

Entrée [57]:

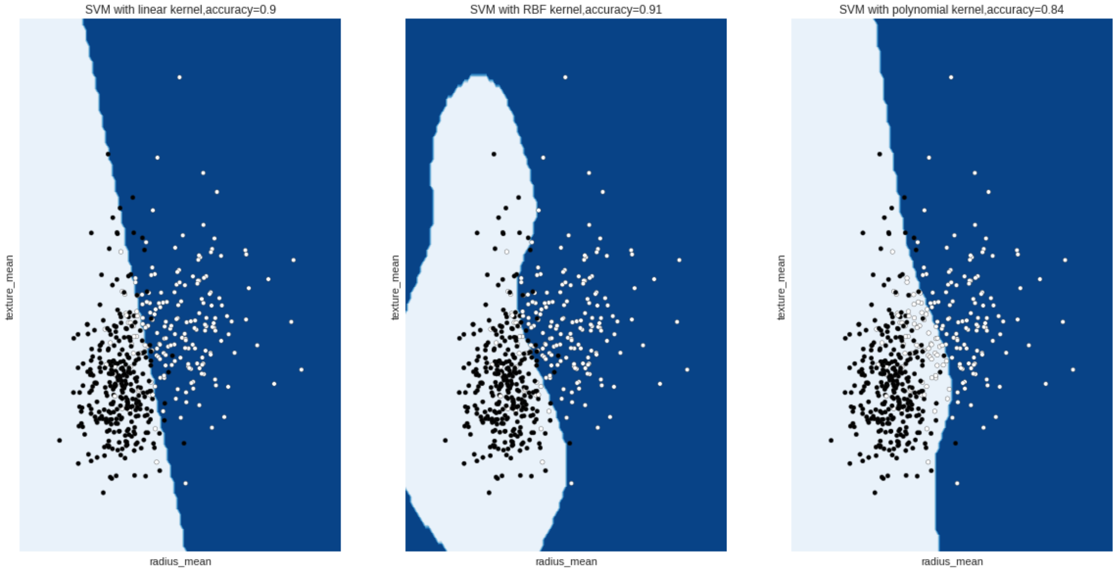
```
SVM = svm.SVC(probability=True)
SVM.fit(x_train, y_train)

# Make predictions on the test set
y_pred = SVM.predict(x_test)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
y_test_binarized = label_binarize(y_test, classes=np.unique(y_t
if len(y_test_binarized[0])<=1:
    fig,ax_roc = plt.subplots()
    RocCurveDisplay.from_estimator(SVM, x_test, y_test, ax=ax_r
    plt.plot([0,1],[0,1],'--',c='black')
    ax_roc.set_title("Receiver Operating Characteristic (ROC) c
    #ax_roc.grid(linestyle="--",c='black')
    plt.legend()
    plt.show()
elif (len(y_test_binarized[0])>1):
    C=["#4B2991","#952EA0","#D44292","#F66D7A","#F6A97A"]
    Classses=np.unique(y_test)
    pred_prob=SVM.predict_proba(x_test)
    fpr = {}
    tpr = {}
    thresh ={}
    roc_auc = dict()
    n_classe = len(Classses)
    for i in range (n_classe):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[
        roc_auc[i] = auc (fpr[i], tpr[i])
    # traçage
        plt.plot(fpr[i], tpr[i],label='%s vs Rest (AUC=%0.2f)'%
    plt.plot([0,1], [0,1],linestyle='--', c='black')
    plt.xlim([0,1])
    plt.ylim([0,1.05])
    plt.title('Receiver Operting Characteristic(ROC) courves')
    plt.xlabel('False Rate')
    plt.ylabel('True Rate')
    plt.legend()
    plt.show()
```


Receiver Operating Characteristic (ROC) curves
True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
SVC (AUC = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0

Comparison des 3 kernel de SVM

Entrée [58]:

```
import matplotlib.pyplot as plt

from sklearn import datasets, svm
from sklearn.inspection import DecisionBoundaryDisplay

XX = X.iloc[:, 0:2]
YY = Y

# we create an instance of SVM and fit out data. We do not scal
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
models = (
    svm.SVC(kernel="linear", C=C),
    svm.SVC(kernel="rbf", gamma=0.7, C=C),
    svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),
)
models = (clf.fit(XX, YY) for clf in models)

# title for the plots
titles = (
    "SVM with linear kernel",
    "SVM with RBF kernel",
    "SVM with polynomial kernel",
)


# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(1, 3,figsize=(20, 10))


#plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = XX.iloc[:, 0], XX.iloc[:, 1]

for clf, title, ax in zip(models, titles, sub.flatten()):
    disp = DecisionBoundaryDisplay.from_estimator(
        clf,
        XX,
        response_method="predict",
        cmap=plt.cm.Blues,
        alpha=1,
        ax=ax,

    )
    m=round(clf.score(XX,YY),2)
    ax.scatter(X0, X1, c=YY, s=20,cmap=plt.cm.gist_gray, edgeco

    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(f"{title},accuracy={m}")

plt.show()
```


SVM with linear kernel,accuracy=0.9
SVM with RBF kernel,accuracy=0.91
SVM with polynomial kernel,accuracy=0.84
texture_mean
radius_mean

Entrée [ ]:

```
1
```