

Direction de l'innovation et des relations avec les entreprises

cnrs **formation**
entreprises

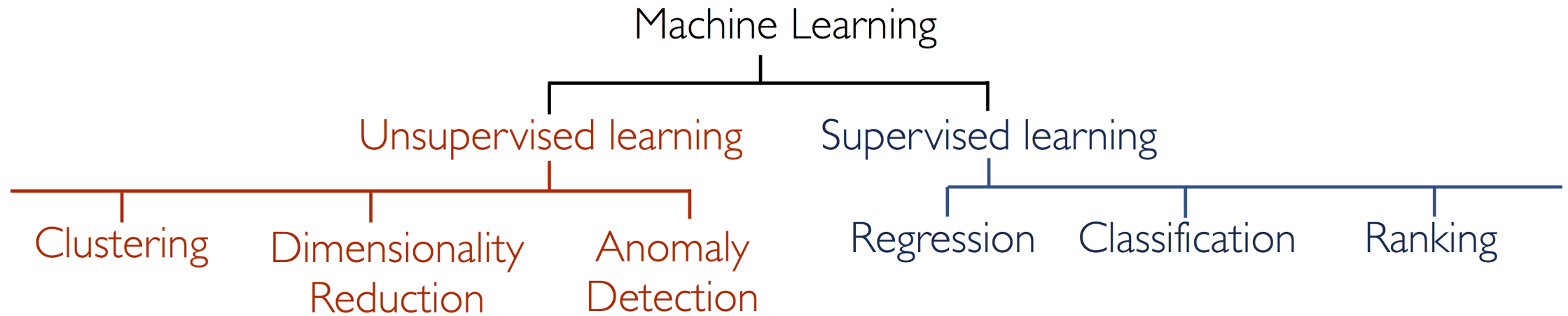
Apprentissage supervisé

Alexandre Boucaud (CNRS/APC)





Préambule

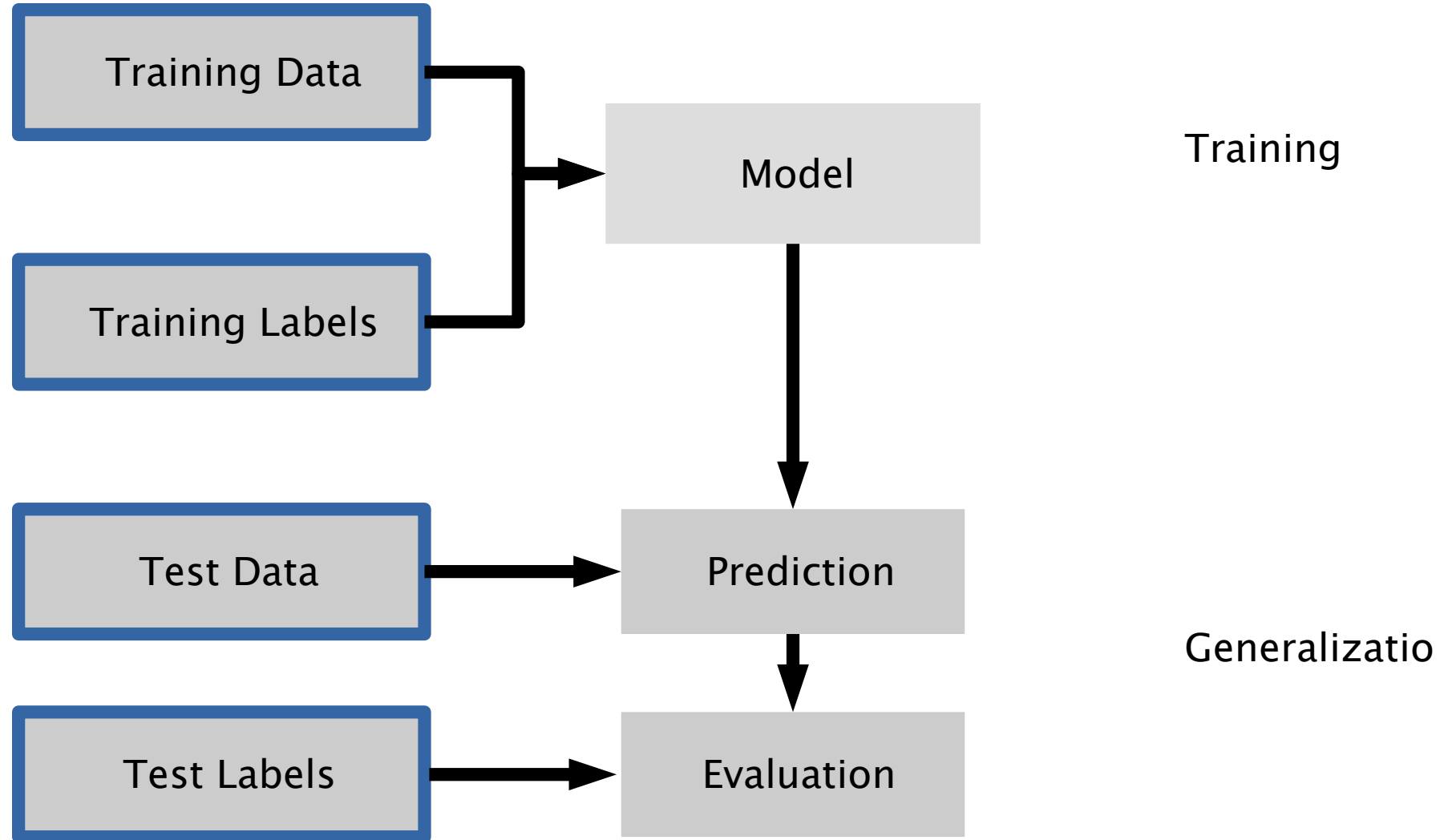




L'apprentissage supervisé consiste à construire un modèle du lien entre deux ensembles de données :

- Les données observées X contenant m exemples à n dimensions
- Une variable externe y , que l'on cherche à prédire, en général appelée **cible** ou **label**.







Rappel : représentation des données dans scikit-learn

- Les données sont représentées par un tableau numpy de dimension 2 (matrice $\mathbb{R}^{m \times n}$):
 - **m** lignes = nombre d'exemples
 - **n** colonnes = nombre de variables (**features** en anglais)

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$





- Les labels associés à chaque exemple sont représentés par un tableau numpy de dimension 1 (matrice $\mathbb{R}^{m \times 1}$)
 - **m** lignes = nombre d'exemples

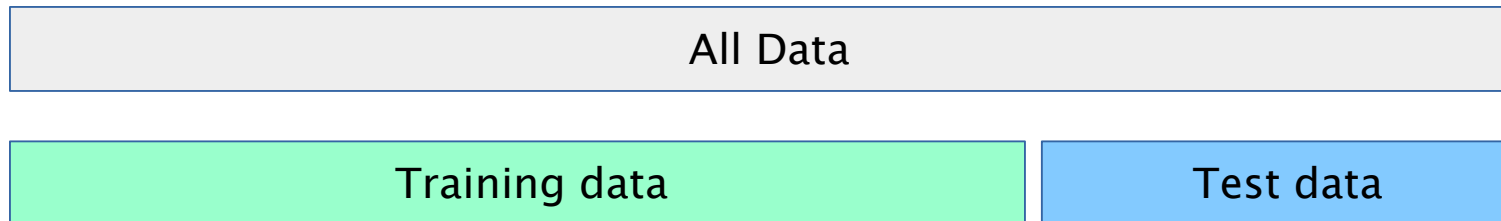
$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} .$$





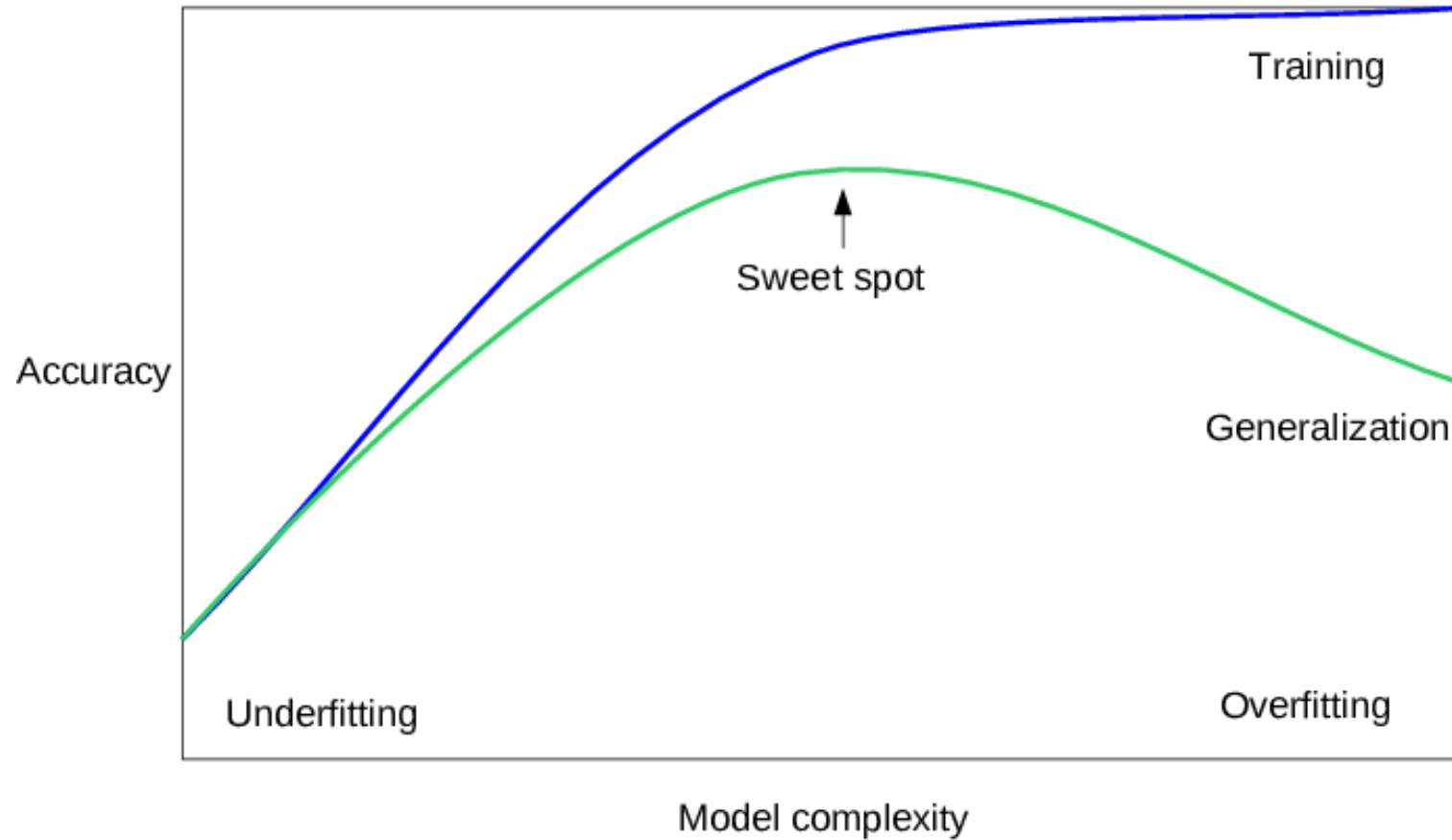
Séparation des données en données de test et d'entraînement

- Apprendre un modèle sur des données et le tester sur les mêmes données est une erreur de méthode
- Il est donc nécessaire de disposer de données qui n'ont pas servi à son entraînement pour jauger des performances du modèle.
- Les données de tests seront utilisées pour comparer les différents modèles enregistrés.





Validation du modèle





L'apprentissage supervisé se scinde en deux grandes catégories :

- L'apprentissage lié aux tâches de régression
 - Linear regression
 - Polynomial regression
 - KNN (K-Nearest-Neighbours regressor)
 - SVR (SVM regressor)
 - Random Forests
 - ...





L'apprentissage supervisé se scinde en deux grandes catégories :

- L'apprentissage lié aux tâches de régression
 - Linear regression
 - Polynomial regression
 - KNN (K-Nearest-Neighbours regressor)
 - SVR (SVM regressor)
 - Random Forests
 - ...





Apprentissage supervisé : Régression

On parle de régression quand on cherche à **prédire** une variable continue.

Exemples :

- espérance de vie en fonction du niveau d'études et du niveau de vie
- réchauffement climatique en fonction de la concentration en gaz à effet de sphère





Génération de données

```
In [2]: from sklearn import datasets

N_SAMPLES = 100
# création d'un jeu de données
X, y, coef = datasets.make_regression(n_samples=N_SAMPLES,
                                     n_features=1,      # nb_features
                                     n_informative=1,   # <= nb_features
                                     noise=10,
                                     coef=True,
                                     random_state=0)

plt.scatter(X, y, marker='.', label='data')
plt.legend(loc='lower right')
plt.xlabel("X")
plt.ylabel("y");
```





Séparation de données d'entraînement et de test

```
In [3]: from sklearn.model_selection import train_test_split

(X_train, X_test,
 y_train, y_test) = train_test_split(X, y, shuffle=True, random_state=42)
```





Import du modèle

```
In [4]: from sklearn.linear_model import LinearRegression
```

https://scikit-learn.org/stable/modules/linear_model.html





Instanciación del modelo

```
In [5]: # Création du modèle  
linreg = LinearRegression()
```





Instanciación del modelo

```
In [5]: # Création du modèle  
linreg = LinearRegression()
```

Entraînement

```
In [6]: # Fit model using train data set  
linreg.fit(X_train, y_train);
```





Instanciación du modèle

```
In [5]: # Création du modèle  
linreg = LinearRegression()
```

Entraînement

```
In [6]: # Fit model using train data set  
linreg.fit(X_train, y_train);
```

Prédiction

```
In [7]: # Predict using test data set  
y_pred = linreg.predict(X_test)
```

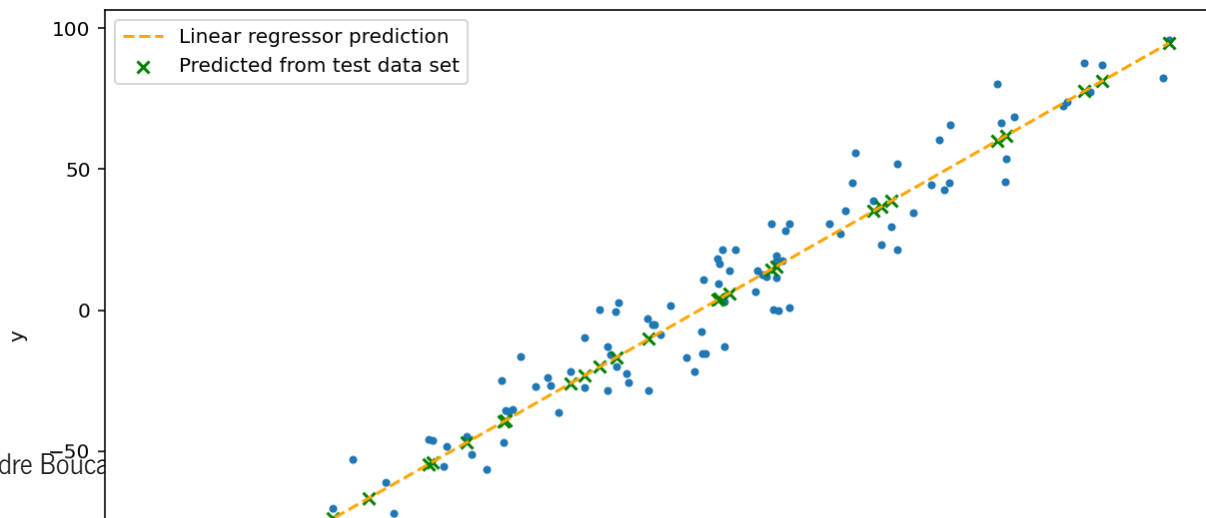




```
In [8]: plt.scatter(X, y, marker='.')
plt.scatter(X_test, y_pred,
            marker='x', color='green',
            label='Predicted from test data set')

X_plot = np.linspace(X.min(), X.max())[:, np.newaxis]
y_plot = linreg.predict(X_plot)
plt.plot(X_plot, y_plot,
         '--', color='orange',
         label='Linear regressor prediction')

plt.legend()
plt.xlabel("X")
plt.ylabel("y");
```





```
In [9]: # Compare estimated coefficient with ground truth
print(f"true    => y = {coef:.4f}*X \npredict => y = {linreg.coef_[0]:.4f}*X")
```

```
true    => y = 42.3855*X
predict => y = 42.3610*X
```





Evaluation de la performance du modèle

L'évaluation de la performance du modèle est fondamentale pour apprécier la qualité de l'apprentissage grâce à des métriques.

Il existe de nombreuses méthodes implémentées dans scikit-learn :
https://scikit-learn.org/stable/modules/model_evaluation.html

On reviendra en détail dessus dans le cours sur l'évaluation et la sélection de modèle





Erreur quadratique moyenne (Mean Squarred Error ou MSE)

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} [y_i - \hat{y}_i]^2.$$

```
In [10]: from sklearn.metrics import mean_squared_error
```

```
MSE = mean_squared_error(y_test, y_pred)
```

```
print(f"MSE = {MSE:0.3f}")
```

```
MSE = 107.718
```





Score R^2

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$





Score R^2

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

```
In [11]: from sklearn.metrics import r2_score
```

```
R2 = r2_score(y_test, y_pred)
```

```
print(f"R2 = {R2:0.3f}")
```

```
R2 = 0.956
```





Score R^2

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

In [11]: `from sklearn.metrics import r2_score`

```
R2 = r2_score(y_test, y_pred)
```

```
print(f"R2 = {R2:0.3f}")
```

R2 = 0.956

In [12]: `accuracy = linreg.score(X_test, y_test)`

```
print(f"accuracy = {accuracy:0.3f}")
```

accuracy = 0.956



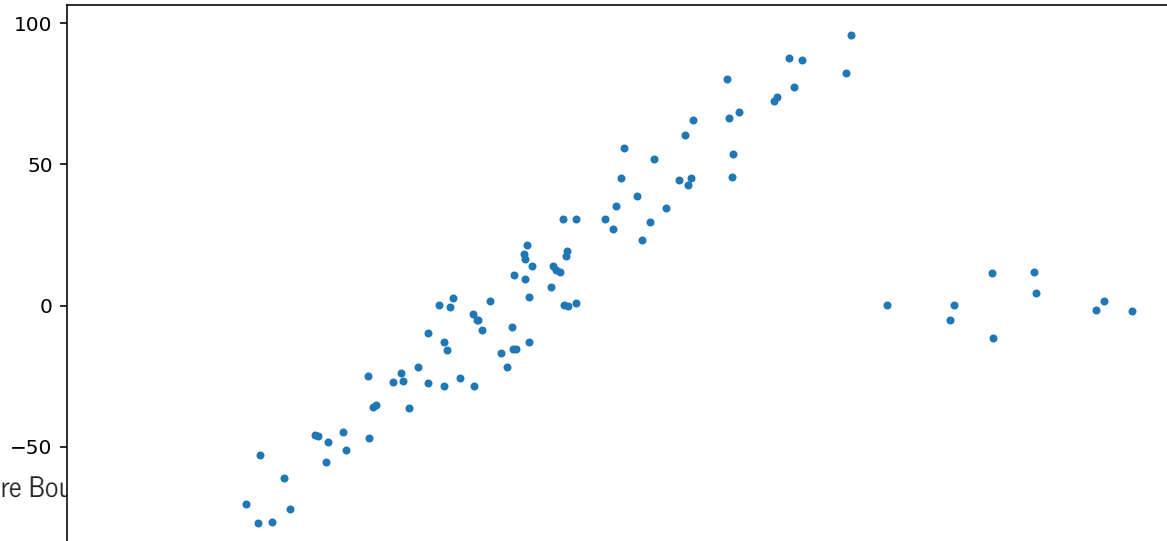


Données de modèles linéaires avec outliers

```
In [13]: # Add outlier data
n_outliers = 10

np.random.seed(0)
X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)
plt.scatter(X, y, marker='.')
```

Out[13]: <matplotlib.collections.PathCollection at 0x134f01cd0>





```
In [14]: (X_train, X_test,  
          y_train, y_test) = train_test_split(X, y, random_state=42)
```





```
In [14]: (X_train, X_test,  
         y_train, y_test) = train_test_split(X, y, random_state=42)
```

```
In [15]: # Fit model using train data set  
linreg.fit(X_train, y_train);
```





```
In [14]: (X_train, X_test,  
         y_train, y_test) = train_test_split(X, y, random_state=42)
```

```
In [15]: # Fit model using train data set  
linreg.fit(X_train, y_train);
```

```
In [16]: # Predict using test data set  
y_pred = linreg.predict(X_test)
```



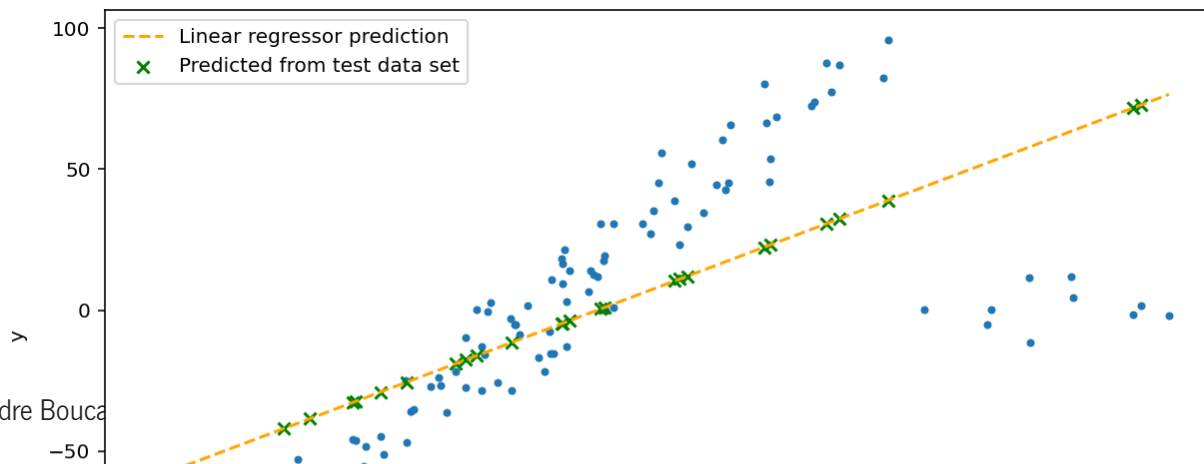


```
In [17]: plt.scatter(X, y, marker='.')
plt.scatter(X_test, y_pred,
            marker='x', color='green',
            label='Predicted from test data set')

X_plot = np.linspace(X.min(), X.max())[:, np.newaxis]
y_plot = linreg.predict(X_plot)

plt.plot(X_plot, y_plot,
         '--', color='orange',
         label='Linear regressor prediction')

plt.legend()
plt.xlabel("X")
plt.ylabel("y");
```





Régression avec l'algorithme RANSAC





Régression avec l'algorithme RANSAC

```
In [18]: from sklearn.linear_model import RANSACRegressor  
ransac = RANSACRegressor()
```





Régression avec l'algorithme RANSAC

```
In [18]: from sklearn.linear_model import RANSACRegressor  
ransac = RANSACRegressor()
```

```
In [19]: # Robustly fit linear model with RANSAC algorithm on train data set  
ransac.fit(X_train, y_train);
```





```
In [20]: # Outlier and inlier RANSAC estimation  
inlier_mask = ransac.inlier_mask_  
outlier_mask = np.logical_not(inlier_mask)
```





```
In [21]: # Predict using test data set  
y_pred_linreg = linreg.predict(X_test)  
y_pred_ransac = ransac.predict(X_test)
```





```
In [22]: data_inlier = plt.scatter(X_train[inlier_mask], y_train[inlier_mask],
                                     color='blue', marker='.',
                                     label='Inliers in train data set')
data_outlier = plt.scatter(X_train[outlier_mask], y_train[outlier_mask],
                            color='red', marker='x',
                            label='Outliers in train data set')

data_pred_linreg = plt.scatter(X_test, y_pred_linreg, marker='o', color='orange')
data_pred_ransac = plt.scatter(X_test, y_pred_ransac, marker='o', color='green')

data_min_max = X_plot = np.linspace(X.min(), X.max())[:, np.newaxis]
data_model_linreg = y_plot_linreg = linreg.predict(X_plot)
data_model_ransac = y_plot_ransac = ransac.predict(X_plot)

plt.plot(X_plot, y_plot_linreg, '--', color='orange', label='Linear regressor')
plt.plot(X_plot, y_plot_ransac, color='green', label='RANSAC linear regressor')

plt.legend()
plt.xlabel("X")
plt.ylabel("y");
```





```
In [23]: # Compare estimated coefficients
print("Estimated coefficients (true, linear regression, RANSAC):")
print(f"true : {coef:.3f}\nlinreg : {linreg.coef_[0]:.3f}\nRANSAC : {ransac.estimator_.coef_[0]:.3f}")
```

Estimated coefficients (true, linear regression, RANSAC):

true : 42.386

linreg : 20.318

RANSAC : 41.366





```
In [23]: # Compare estimated coefficients
print("Estimated coefficients (true, linear regression, RANSAC):")
print(f"true : {coef:.3f}\nlinreg : {linreg.coef_[0]:.3f}\nRANSAC : {ransac.estimator_.coef_[0]:.3f}")
```

```
Estimated coefficients (true, linear regression, RANSAC):
true : 42.386
linreg : 20.318
RANSAC : 41.366
```

Certains modèles sont **par défaut** bien plus résistants aux outliers.





Régression polynomiale

```
In [24]: from sklearn.preprocessing import PolynomialFeatures
```





Régression polynomiale

```
In [24]: from sklearn.preprocessing import PolynomialFeatures
```

```
In [25]: x = np.arange(5).reshape(-1, 1)

poly = PolynomialFeatures(degree=4)

print(poly.fit_transform(x))
```

```
[[ 1.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  2.  4.  8. 16.]
 [ 1.  3.  9. 27. 81.]
 [ 1.  4. 16. 64. 256.]]
```





Régression polynomiale

```
In [24]: from sklearn.preprocessing import PolynomialFeatures
```

```
In [25]: x = np.arange(5).reshape(-1, 1)

poly = PolynomialFeatures(degree=4)

print(poly.fit_transform(x))
```

```
[[ 1.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  2.  4.  8. 16.]
 [ 1.  3.  9. 27. 81.]
 [ 1.  4. 16. 64. 256.]]
```

$[x]$ a été transformé en $[1, x, x^2, x^3, x^4]$ et on peut maintenant utiliser un modèle linéaire pour estimer les coefficients de chaque degré du polynôme.





Génération des données

```
In [26]: # Fonction à approximer par interpolation polynomiale
def f(x):
    return x * np.sin(x)

# On génère les points de données
X = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
rng.shuffle(X)
X = np.sort(X[:40])[:, np.newaxis]

# calculate return values and add some noise
noise = np.random.normal(0, .7, X.shape)
y = f(X) + noise
```





Séparation de données entraînement et données de test

```
In [27]: (X_train, X_test,  
          y_train, y_test) = train_test_split(X, y, random_state=42)
```





Génération des variables polynomiales et entraînement du modèle

```
In [28]: # Fit model using train data set  
poly = PolynomialFeatures(degree=3, include_bias=False)  
X_poly = poly.fit_transform(X_train)
```





Génération des variables polynomiales et entraînement du modèle

```
In [28]: # Fit model using train data set  
poly = PolynomialFeatures(degree=3, include_bias=False)  
X_poly = poly.fit_transform(X_train)
```

```
In [29]: model = LinearRegression()  
model.fit(X_poly, y_train);
```





Génération des variables polynomiales et entraînement du modèle

```
In [28]: # Fit model using train data set  
poly = PolynomialFeatures(degree=3, include_bias=False)  
X_poly = poly.fit_transform(X_train)
```

```
In [29]: model = LinearRegression()  
model.fit(X_poly, y_train);
```

```
In [30]: # Predict using test data set  
y_pred = model.predict(poly.transform(X_test))
```





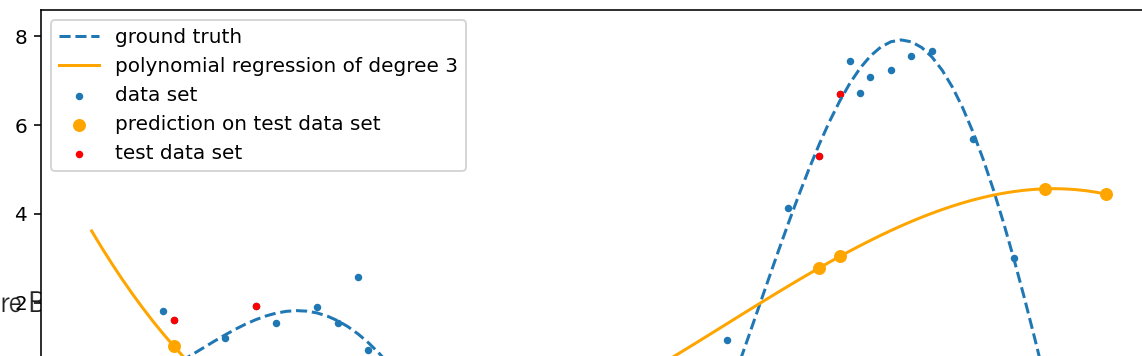
```
In [31]: # notre modèle attendu
X_plot = np.linspace(0, 10, 100)[: , np.newaxis]
plt.plot(X_plot, f(X_plot), ls='--', label="ground truth")

# le jeu de données initial
plt.scatter(X, y, s=30, marker='.', label="data set")

# les données prédites par le modèle
plt.scatter(X_test, y_pred, s=30, color='orange', marker='o', label="prediction on test data set")
plt.scatter(X_test, y_test, s=30, color='r', marker='.', label="test data set")

# le modèle obtenu
y_plot = model.predict(poly.transform(X_plot))
plt.plot(X_plot, y_plot, color='orange',
         label="polynomial regression of degree {}".format(poly.get_params()['degree']))

plt.legend();
```





Enchainement des opérations dans un pipeline

```
In [32]: from sklearn.pipeline import make_pipeline

model = make_pipeline(PolynomialFeatures(3, include_bias=False),
                      LinearRegression())
```





Enchaînement des opérations dans un pipeline

```
In [32]: from sklearn.pipeline import make_pipeline

model = make_pipeline(PolynomialFeatures(3, include_bias=False),
                      LinearRegression())
```

Equivalent à ce qui a été fait juste au dessus.

`model` se comporte comme un modèle de `LinearRegression()` mais les valeurs en entrée sont d'abord transformées grâce au `PolynomialFeatures`. De cette manière, **plus besoin des étapes intermédiaires** de bookkeeping : `X_trans..`





Il est donc maintenant très facile de tester certaines hypothèses de travail comme par exemple le degré maximal du polynôme :

```
In [33]: models = [make_pipeline(PolynomialFeatures(degree, include_bias=False),  
                                LinearRegression())  
                  for degree in (3, 5, 7)]
```





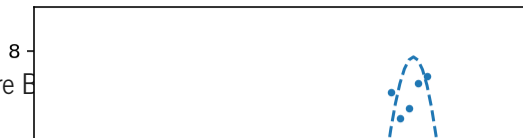
```
In [34]: fig, axes = plt.subplots(1, 3, figsize=(15, 8))

for n, model in enumerate(models):
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_plot = model.predict(X_plot)

    axes[n].plot(X_plot, f(X_plot), ls='--', label="ground truth")
    axes[n].scatter(X, y, s=30, marker='.', label="data set")

    axes[n].scatter(X_test, y_pred, s=30,
                    marker='o', color='orange',
                    label="prediction on test data set")
    axes[n].scatter(X_test, y_test, s=30,
                    marker='.', color='red',
                    label="test data set")
    axes[n].plot(X_plot, y_plot, color='orange',
                 label=("polynomial regression of degree "
                        f"{model.get_params()['polynomialfeatures__degree']}"))
axes[1].legend();
```





Régression K-Nearest Neighbor (KNN)

```
In [35]: from sklearn.neighbors import KNeighborsRegressor
```





```
In [36]: n_neighbors = 2  
weights = 'uniform'  #'distance'  
knn = KNeighborsRegressor(n_neighbors, weights=weights)
```

```
In [37]: knn.fit(X_train, y_train);
```

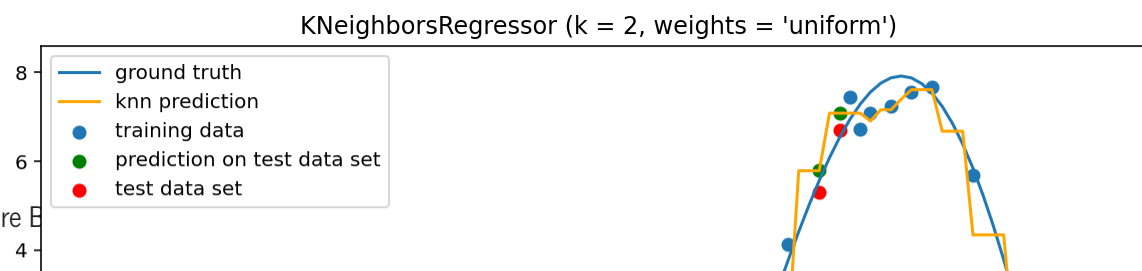
```
In [38]: y_pred = knn.predict(X_test)
```





```
In [39]: X_plot = np.linspace(0, 10, 100)[: , np.newaxis]
y_plot = knn.predict(X_plot)
plt.scatter(X_train, y_train,
            label='training data')
plt.plot(X_plot, f(X_plot),
         label="ground truth")
plt.scatter(X_test, y_pred,
            color='green',
            label='prediction on test data set')
plt.scatter(X_test, y_test,
            color='red',
            label='test data set')
plt.plot(X_plot, y_plot,
         color='orange',
         label='knn prediction')
plt.legend()
plt.title(f"KNeighborsRegressor (k = {n_neighbors}, weights = '{weights}')
```

Out[39]: Text(0.5, 1.0, "KNeighborsRegressor (k = 2, weights = 'uniform')")





```
In [40]: n_neighbors = [1, 5, 10]
weights = ['uniform', 'distance']

fig, axes = plt.subplots(len(n_neighbors), len(weights), figsize=(15, 10))

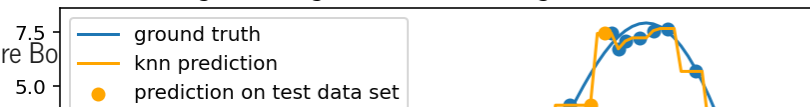
for i, n in enumerate(n_neighbors):
    for j, w in enumerate(weights):
        knn = KNeighborsRegressor(n, weights=w)

        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)

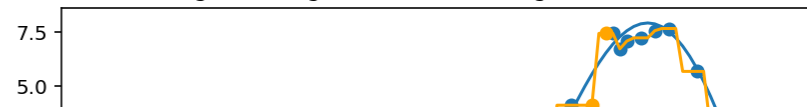
        X_plot = np.linspace(0, 10, 100)[: , np.newaxis]
        y_plot = knn.predict(X_plot)

        axes[i,j].scatter(X_train, y_train)
        axes[i,j].plot(X_plot, f(X_plot), label="ground truth")
        axes[i,j].scatter(X_test, y_pred, label='prediction on test data set', color='orange')
        axes[i,j].plot(X_plot, y_plot, label='knn prediction', color='orange')
        axes[i,j].set_title("KNeighborsRegressor (k = %i, weights = '%s')" % (n, w));+6
axes[0,0].legend();
```

KNeighborsRegressor (k = 1, weights = 'uniform')



KNeighborsRegressor (k = 1, weights = 'distance')





Apprentissage supervisé : Classification

Les exemples appartiennent à plusieurs classes et on veut apprendre à un modèle à partir de données déjà labélisées à prédire la classe de données non labélisées.

exemples:

- spam ou non spam
- Tumeur bénigne ou maligne





Génération de données

```
In [41]: from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=400, centers=2,
                  random_state=0, cluster_std=1)

print('X ~ n_samples x n_features:', X.shape)
print('y ~ n_samples:', y.shape)

print('\nFirst 5 samples:\n', X[:5, :])
print('\nFirst 5 labels:', y[:5])
```

```
X ~ n_samples x n_features: (400, 2)
y ~ n_samples: (400,)
```

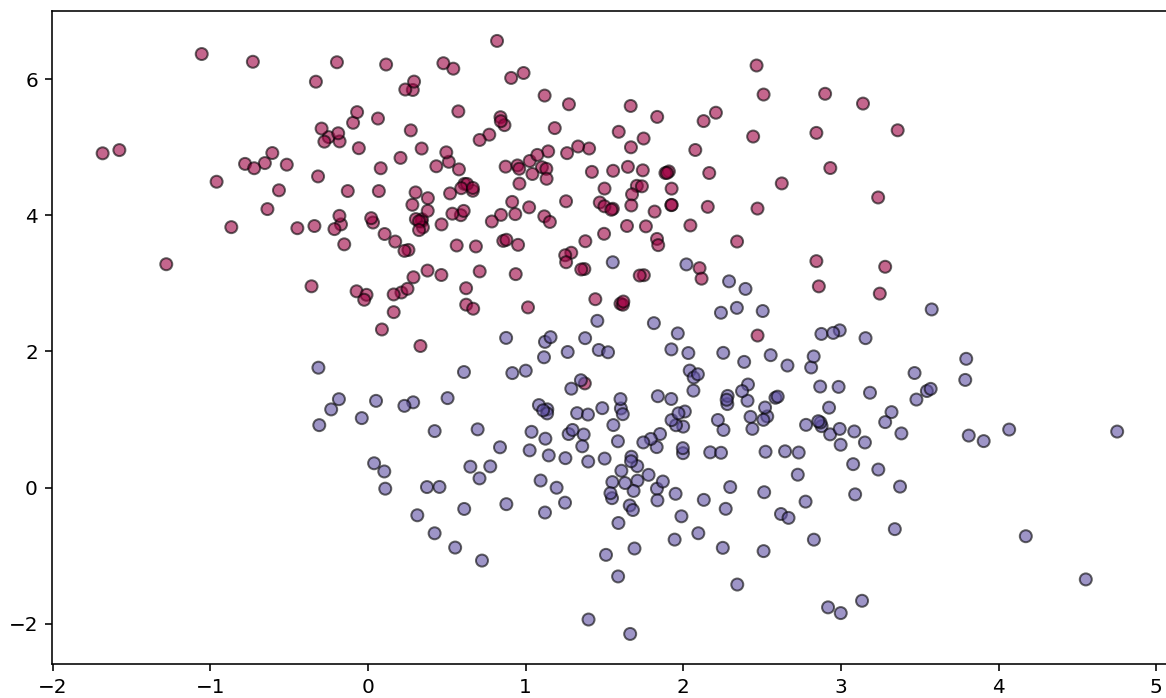
```
First 5 samples:
[[ 0.71  0.14]
 [ 2.83 -0.76]
 [ 1.71  0.32]
 [ 2.53  1.05]
 [ 0.42  0.83]]
```

```
First 5 labels: [1 1 1 1 1]
```





```
In [42]: plt.scatter(X[:, 0], X[:, 1],  
                    c=y, cmap=plt.cm.Spectral,  
                    edgecolors='black', alpha=0.6);
```





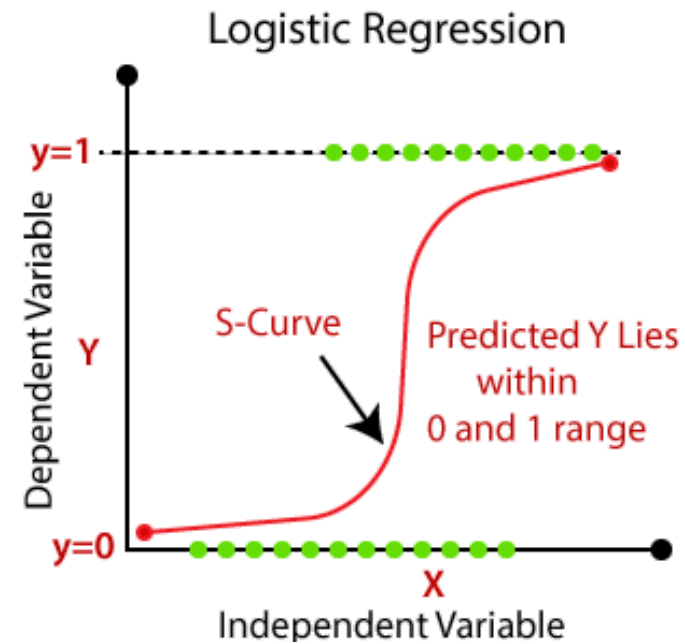
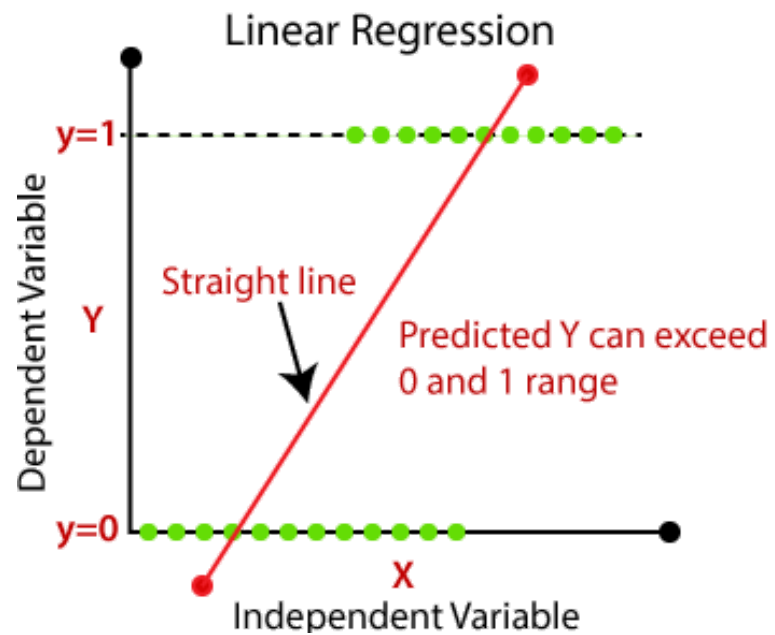
Séparation des données entre données d'entraînement et données de test

```
In [43]: (X_train, X_test,
          y_train, y_test) = train_test_split(X, y,
                                              test_size=0.25,
                                              random_state=1234,
                                              stratify=y)
```





Régression logistique





```
In [44]: from sklearn.linear_model import LogisticRegression
```

```
logistic_regressor = LogisticRegression()
```



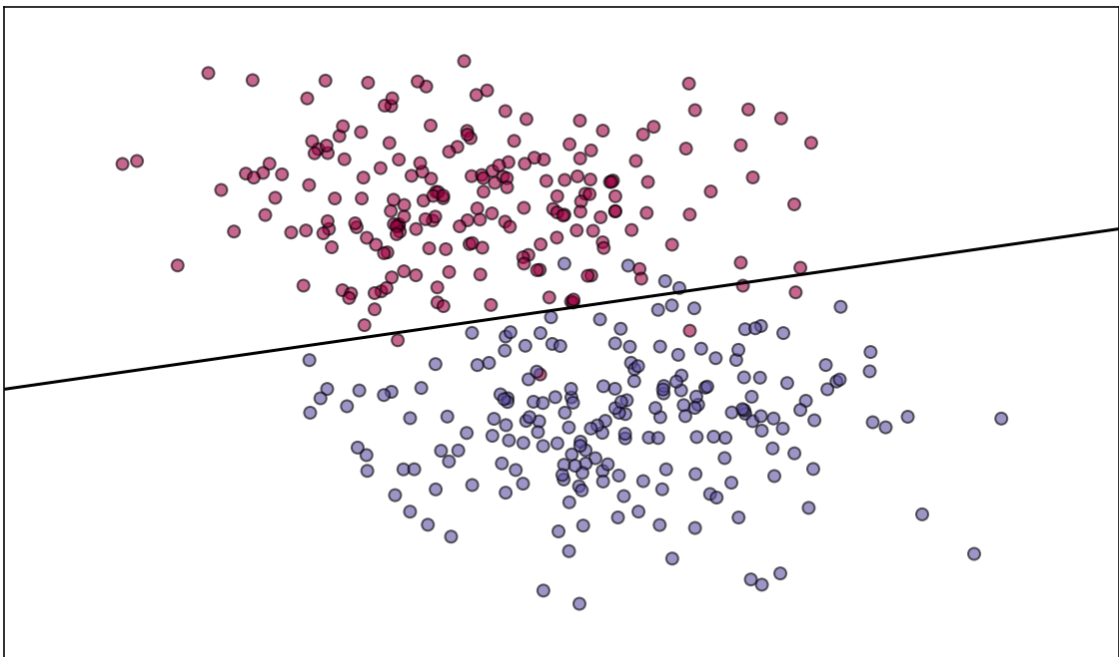


```
In [45]: logistic_regressor.fit(X_train, y_train);
```





```
In [47]: ax = plt.axes()
ax.scatter(X[:, 0], X[:, 1], c=y,
           cmap=plt.cm.Spectral,
           edgecolors='black', alpha=0.6)
plot_2d_separator(logistic_regressor, X)
plt.show()
```





```
In [48]: print(logistic_regressor.coef_)
```

```
[[ 0.95 -3.  ]]
```





Evaluation de la performance du modèle

Comme pour la régression, il existe de nombreuses méthodes d'évaluation de modèles pour la classification.

On en verra un certain nombre dans le cours sur l'évaluation et la sélection de modèle.

https://scikit-learn.org/stable/modules/model_evaluation.html





```
In [49]: y_pred = logistic_regressor.predict(X_test)
```

```
In [50]: print(y_pred, '\n')  
print(y_test)
```

```
[0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0  
 1 0 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1  
 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 0 0]
```

```
[0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0  
 1 0 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1  
 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 0 0]
```





Accuracy

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

```
In [51]: # Return the mean accuracy on the given test data and labels.  
from sklearn.metrics import accuracy_score  
  
accuracy_score(y_pred, y_test)
```

Out[51]: 0.98

```
In [52]: np.mean(y_pred == y_test)
```

Out[52]: 0.98

```
In [53]: logistic_regressor.score(X_test, y_test)
```

Out[53]: 0.98





K Nearest Neighbors (KNN)





```
In [54]: from sklearn.neighbors import KNeighborsClassifier
```





```
In [54]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [55]: knn = KNeighborsClassifier(n_neighbors=7)
```



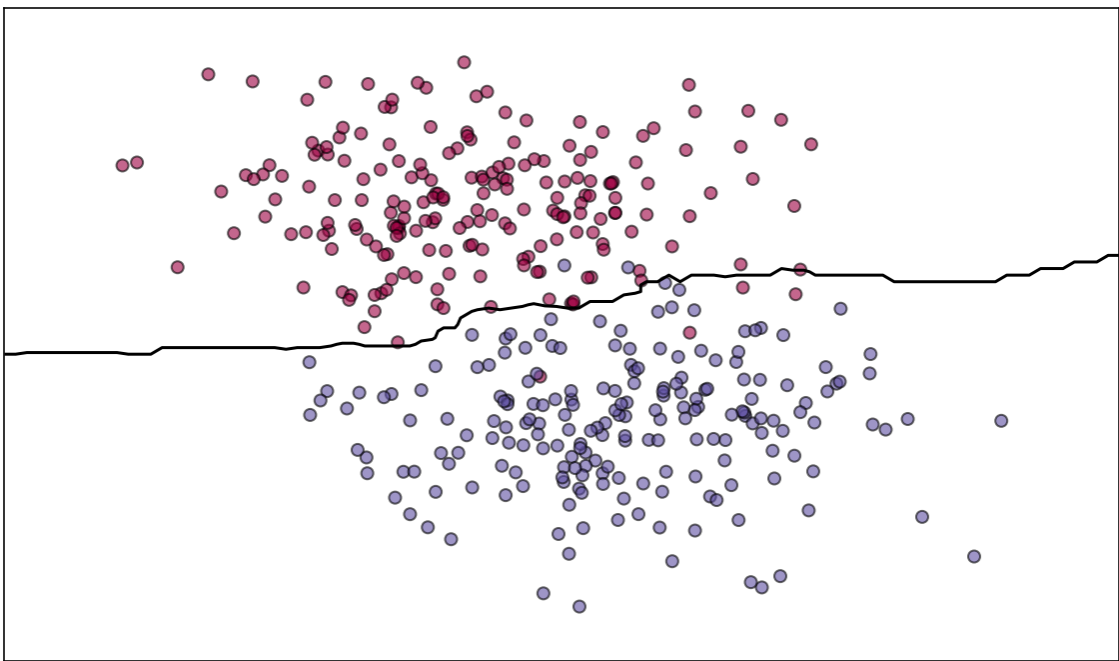


```
In [56]: knn.fit(X_train, y_train);
```





```
In [57]: ax = plt.axes()  
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral ,edgecolors='black', alpha=0.6)  
plot_2d_separator(knn, X);
```



```
In [58]: knn.score(X_test, y_test)
```

Out[58]: 0.98

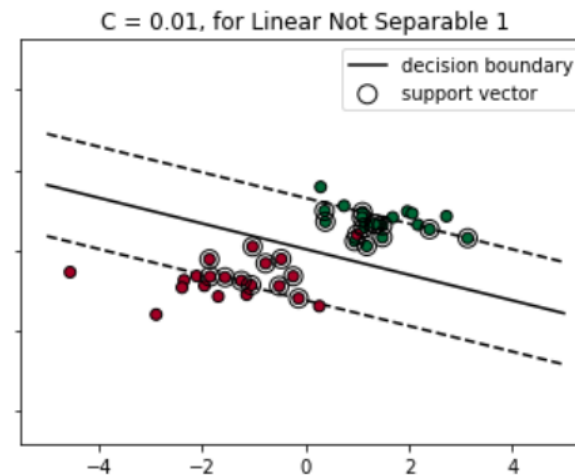
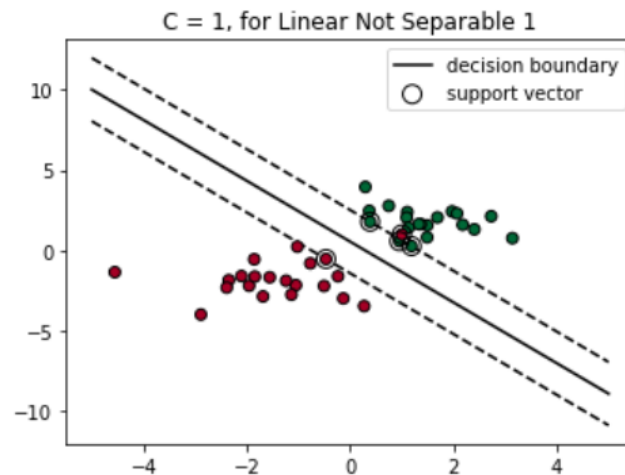




Support Vector Machines (SVM)

<https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496>

```
In [59]: from sklearn.svm import SVC  
  
# SVC = Support Vector Classifier
```





```
In [61]: def plot_svm_separation(clf, X, y):
xx = np.linspace(-1, 5, 10)
yy = np.linspace(-1, 5, 10)

X1, X2 = np.meshgrid(xx, yy)
Z = np.empty(X1.shape)
for (i, j), val in np.ndenumerate(X1):
    x1 = val
    x2 = X2[i, j]
    p = clf.decision_function([[x1, x2]])
    Z[i, j] = p[0]

ax = plt.axes()
ax.contour(X1, X2, Z, [-1.0, 0.0, 1.0],
           colors='k', linestyles=['dashed', 'solid', 'dashed'])
ax.scatter(X[:, 0], X[:, 1], c=y,
           cmap=plt.cm.Spectral,
           edgecolors='black', alpha=0.6);
```

```
In [62]: clf = SVC(C=0.1, kernel='poly')
clf.fit(X_train, y_train)
plot_svm_separation(clf, X, y)
```





Cas de données non linéairement séparable

```
In [63]: np.random.seed(0)
N = 200      # number of points per class
D_in = 2     # dimensionality
D_out = 3    # number of classes
X = np.zeros((N*D_out, D_in))
y = np.zeros(N*D_out, dtype='uint8')
for j in range(D_out):
    ix = range(N*j, N*(j+1))
    r = np.linspace(0.0, 1, N) # radius
    t = np.linspace(j*4, (j+1)*4, N) + np.random.randn(N)*0.2 # theta
    X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
    y[ix] = j
Y = np.atleast_2d(y).T.astype(float) # Vector made out of y, for convenience
```

```
In [64]: fig = plt.figure(figsize=(10, 10))
plt.scatter(X[:, 0], X[:, 1], c=y, s=40,
            cmap=plt.cm.Spectral, alpha=0.6,
            edgecolors='black', linewidths=0.2)
plt.axis('equal') ; plt.xlim([-1.5, 1.5]) ; plt.ylim([-1.5, 1.5]) ;
```





Régression logistique

```
In [65]: clf = LogisticRegression()  
  
         clf.fit(X,Y)
```





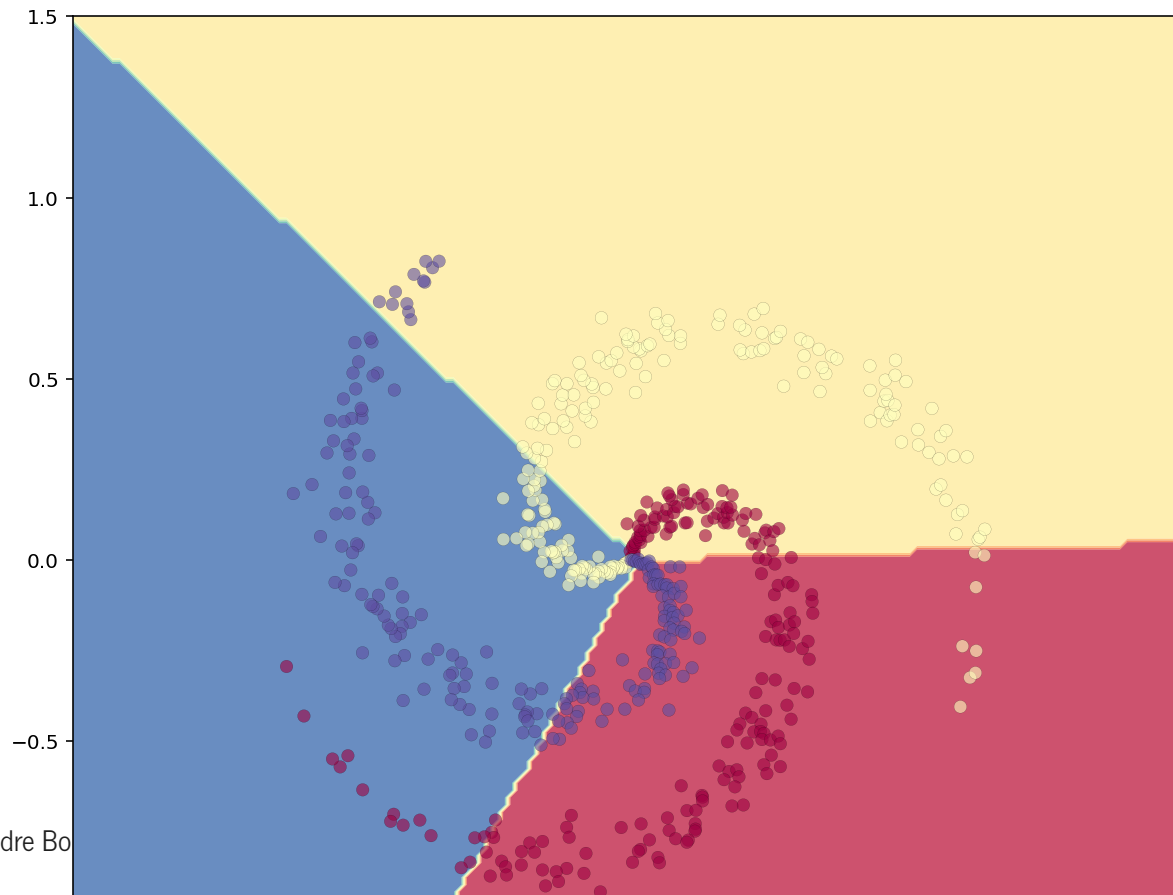
```
In [67]: # Prepare a grid for plotting contours  
step = 0.02  
x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1  
y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1  
xi, yi = np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))  
X_plot = np.c_[xi.ravel(), yi.ravel()]
```





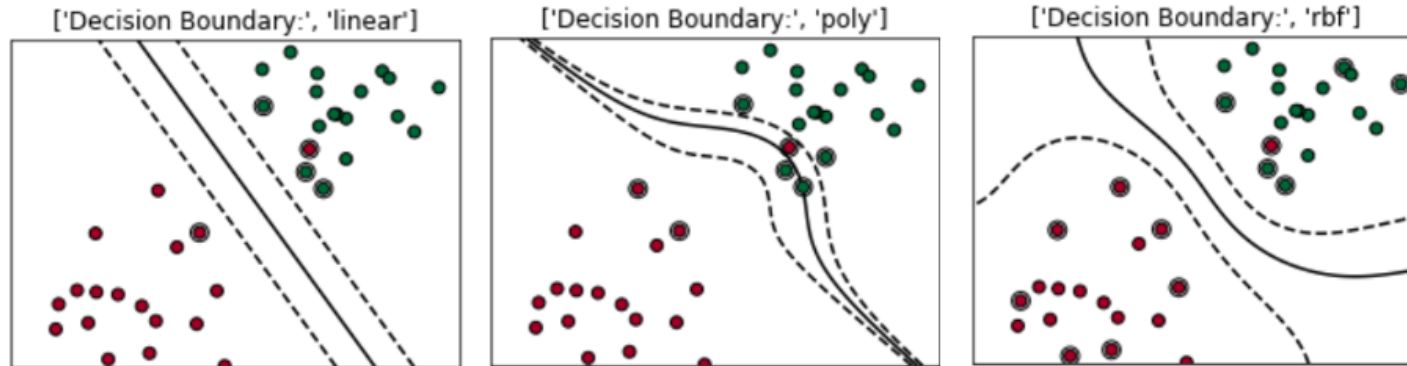
```
In [68]: Z = clf.predict(X_plot).reshape(xi.shape)
```

```
fig = plt.figure(figsize=(10, 10))  
plt.contourf(xi, yi, Z, cmap=plt.cm.Spectral, alpha=0.8, antialiased=True)  
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral, alpha=0.6  
            , edgecolors='black', linewidths=0.1)  
plt.axis('equal'); plt.xlim(-1.5, 1.5); plt.ylim(-1.5, 1.5) ;
```





SVMs avec différents kernels





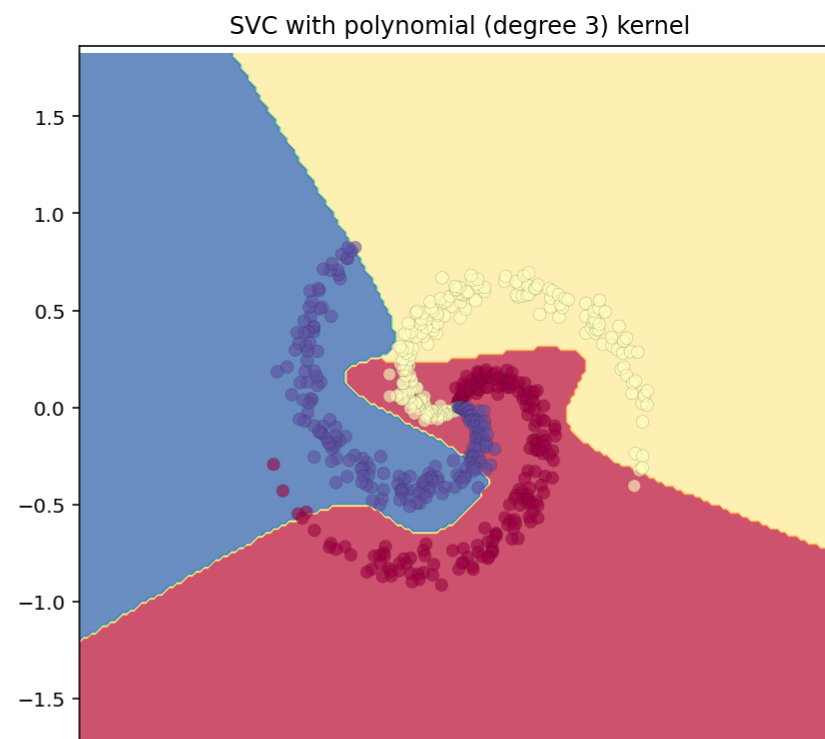
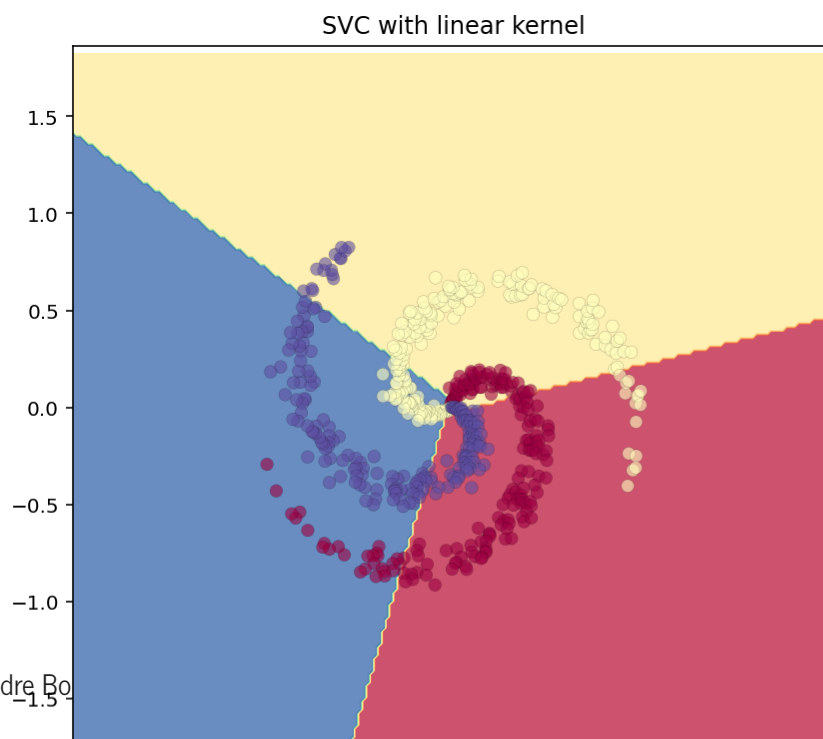
```
In [69]: # Define models
C = 100.0 # SVM regularization parameter
models = (SVC(kernel='linear', C=C),
          SVC(kernel='poly', degree=3, C=C),
          SVC(kernel='poly', degree=5, C=C),
          SVC(kernel='rbf', gamma='auto', C=C)) # N.B.: "RBF" = "Gaussian kernel" try gamma = 10.
models = (clf.fit(X, y) for clf in models)

# Define model titles
titles = ('SVC with linear kernel',
         'SVC with polynomial (degree 3) kernel',
         'SVC with polynomial (degree 5) kernel',
         'SVC with RBF kernel')
```





```
In [70]: # Plot resulting classifiers
fig, axes = plt.subplots(2, 2, figsize=(15, 15))
for ax, clf, title in zip(axes.flat, models, titles):
    Z = clf.predict(X_plot).reshape(xi.shape)
    ax.contourf(xi, yi, Z, cmap=plt.cm.Spectral, alpha=0.8)
    ax.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral, alpha=0.6
               , edgecolors='black', linewidths=0.1)
    ax.axis("equal") ; plt.xlim(-1.5, 1.5); plt.ylim(-1.5, 1.5) ;
    ax.set_title(title)
plt.show()
```





Conclusion

Les algorithmes d'apprentissage supervisé permettent d'apprendre à partir d'un jeu de données d'entraînement (**X_{train} , y_{train}**) afin de prédire une valeur continue (regression) ou une classe (classification) pour de **nouvelles entrées X** .

