Matière : **Machine Learning**

DSI 3

Chapitre V: La classification (Decision Tree et Random Forest,...)

Enseignantes :

Naïma Halouani
Hounaïda Moalla
ISET Sfax

Objectif

L'étudiant découvrira :

- la classification des arbres de décision,

- les mesures de sélection d'attributs

- et comment créer et optimiser le classificateur d'arbres de décision à l'aide du package Python Scikit-learn.

- Extension du DT avec l'algorithme Random Forest

- L'implémentation du SVM et Logistic Regression

Plan

- Decision Tree
  - Définition
  - Principe
  - Traduction en Python
- Random Forest
- SVM
- Logistic Regression

Decision Tree

Définition

L'arbre de décision est un algorithme ML dont le **temps d'apprentissage** est plus rapide par rapport à d'autres algorithmes.

La **complexité temporelle** des arbres de décision est fonction du nombre d'enregistrements et du nombre d'attributs (features) dans les données.

Les arbres de décision peuvent traiter des données de grande dimension avec une bonne précision.

Decision Tree

Définition

Un arbre de décision est une structure arborescente de type organigramme où :

- **un nœud interne** représente une caractéristique (ou un attribut),

- **la branche** représente une règle de décision et chaque nœud feuille représente le résultat.

- Le nœud le plus haut dans un arbre de décision est appelé **nœud racine**.

- Le choix de l'attribut à mettre à la racine de l'arbre (**nœud racine**) puis de chacun des nœuds fils se fait en calculant soit **l'indice Gini** soit **l'entropie** correspondante.

Decision Tree

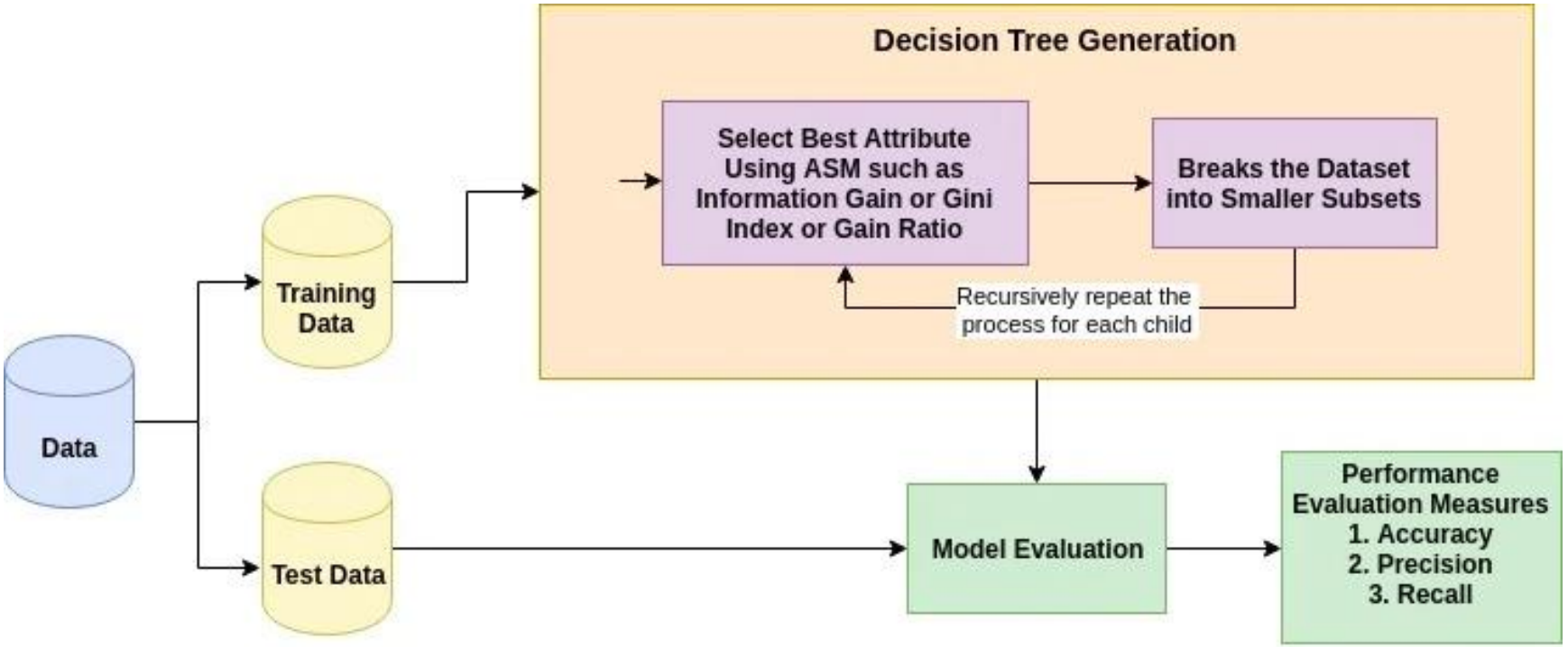
Principe

Le procès complet de construction de l'arbre de décision

1. À l'initialisation, l'arbre est totalement vide.
2. Le score de Gini de toutes les décisions possibles est calculé.
3. La décision qui présente le score Gini maximal est choisie comme racine (la nœud le plus pur)
4. Tant qu'il est possible de faire un split et que le critère d'arrêt n'est pas respecté
5. Pour chaque décision qu'il est possible d'ajouter à l'arbre; Faire 6.
6. Calcul du score Gini de la décision courante
7. Sélection de la décision admettant le score max et ajout de celle-ci à l'arbre

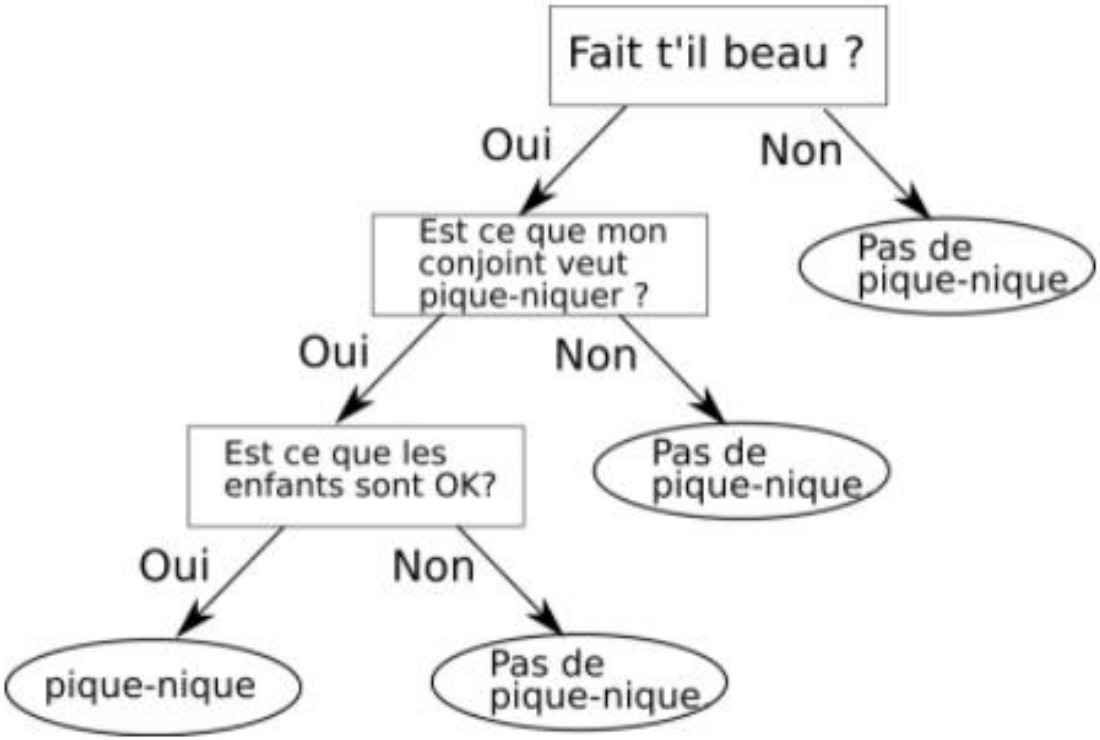
Decision Tree

Principe



Decision Tree

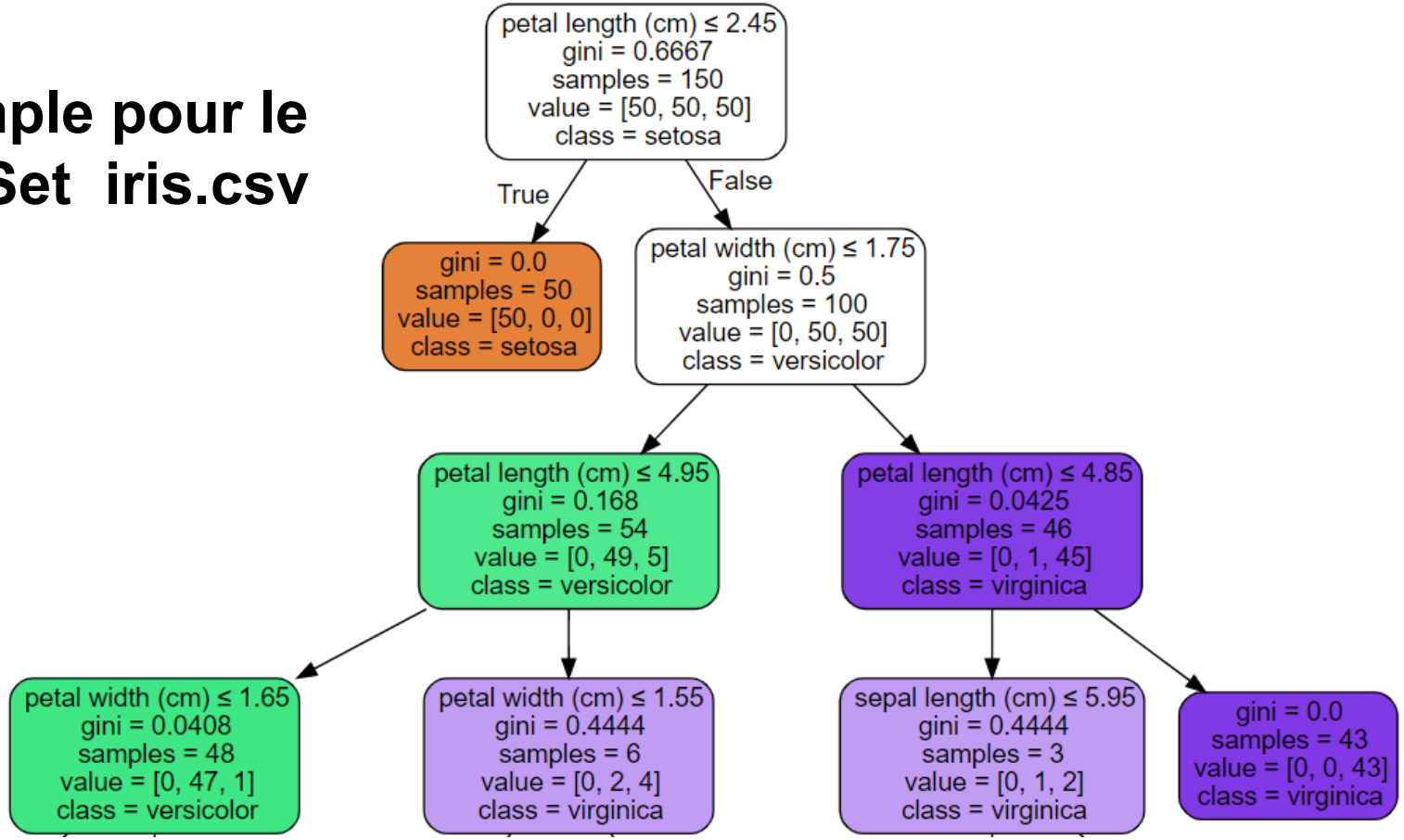
Exemple



Decision Tree

Exemple

Exemple pour le DataSet iris.csv



Decision Tree

Interprétation

- **gini** - c'est la valeur de l'impureté gini. Il représente la pureté d'un nœud.
- C'est la fonction de coût que nous chercherons pour déterminer la qualité de nos divisions.
- Un coefficient de gini égale à 0 représente un nœud pur.
- Le coefficient de Gini est calculé suivant la formule:

$$Gini = \sum (pk \times (1 - pk))$$

Soit S un échantillon et S1,...,Sk la partition de S selon les classes de l'attribut cible. On définit pk=Si/S

Decision Tree

Interprétation

- **Samples** : Il s'agit du nombre d'instances d'apprentissage dans le nœud. À partir de l'image, la valeur d'échantillon au nœud racine est l'ensemble de données complet (150). De plus, la somme des valeurs d'échantillon des nœuds enfants doit être égale à celle du nœud parent. Par exemple. 50 + 100 = 150.

- **value** : Combien d'instances d'apprentissage de chaque classe se trouve dans ce nœud. Ces valeurs doivent résumer la valeur des échantillons (samples) de ce nœud.

- **classe** : **classe** prédite de ce nœud.

Decision Tree

Traduction en Python

1 **Apprentissage - Prediction**

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()

dt.fit(X_train,y_train)

y_pred_dt = dt.predict(X_test)

for i in range(10):
    print(y_test[i],y_pred_dt[i])
```

Decision Tree

Traduction en Python

2 Evaluation

```
print('Confusion matrix dt \n', confusion_matrix(y_test,y_pred_dt))

print('Accuracy dt', accuracy_score(y_test,y_pred_dt))

print('Recall dt', recall_score(y_test,y_pred_dt))

print('Precision dt', precision_score(y_test,y_pred_dt))
```

```
print(classification_report(y_test,y_pred_dt))
```

Decision Tree

Traduction en Python

3 Visualisation

```
!pip install graphviz
```

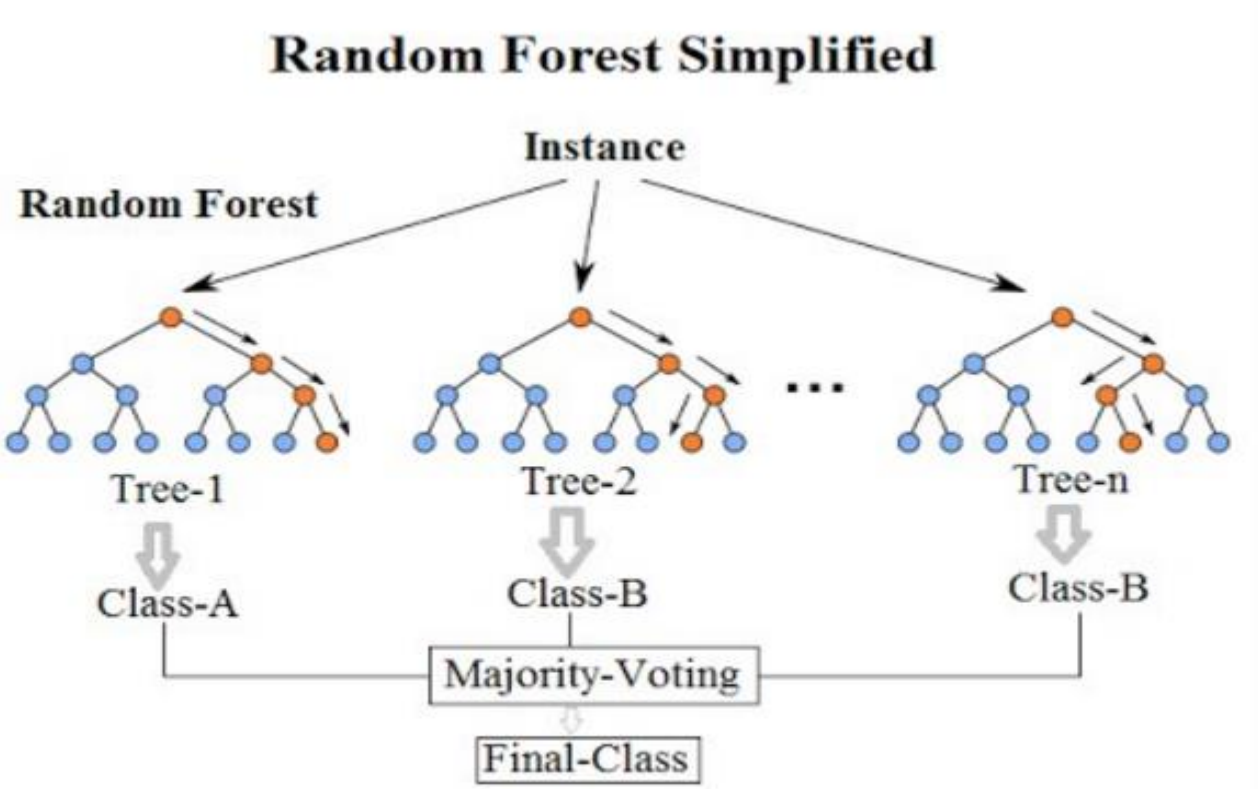
```
import graphviz
from sklearn import tree
from sklearn.tree import export_graphviz
model = DecisionTreeClassifier(max_depth = 5)
model.fit(X,y)
```

```
tree.export_graphviz(model,feature_names = features,\
                     out_file = 'nom_image.dot',\
                     label = 'all',\
                     filled = True,\
                     rounded = True)
```

➡ Consulter le contenu du fichier ‘nom_image.dot’

Random Forest

Random Forest est une extension de l'algorithme des arbres de décision, conçu pour améliorer leur performance et leur robustesse.



Random Forest

Traduction en Python

1 **Apprentissage - Prediction**

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=500)

rf.fit(X_train,y_train)

y_pred_rf = rf.predict(X_test)
```

Random Forest

Traduction en Python

2 Evaluation

```
print('Confusion matrix rf \n', confusion_matrix(y_test,y_pred_rf))

print('Accuracy rf', accuracy_score(y_test,y_pred_rf))

print('Recall rf', recall_score(y_test,y_pred_rf))

print('Precision rf', precision_score(y_test,y_pred_rf))
```

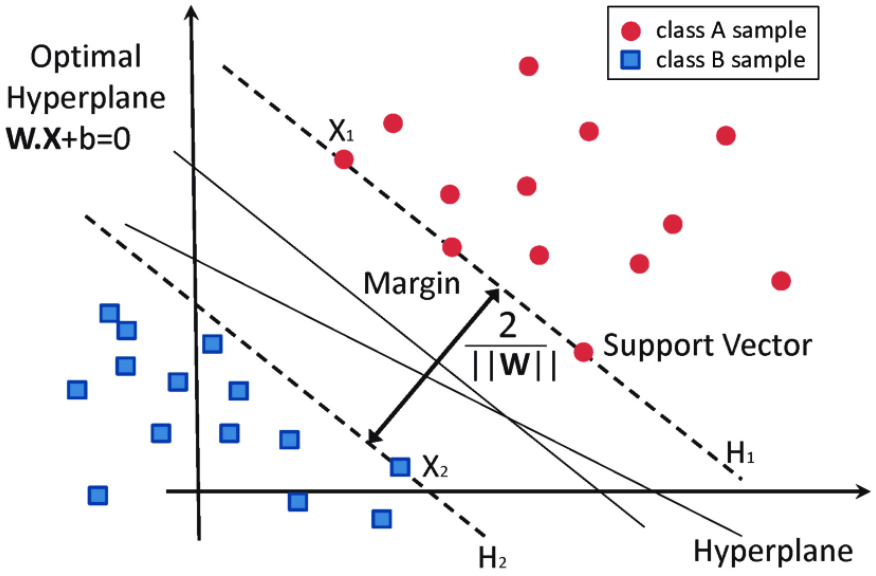
```
print(classification_report(y_test,y_pred_rf))
```

SVM

Linear SVM

Principe général

- Tracer une frontière : Il cherche une ligne (ou un hyperplan) qui sépare les classes.
- Maximiser la marge : Il choisit la ligne qui est la plus éloignée possible des points de chaque classe.
- Utilisation de "vecteurs de support" : Ce sont les points les plus proches de la frontière de séparation, qui influencent sa position.



SVM

Linear SVM

Exemple simple :

Imaginons que tu veux séparer des pommes et des oranges sur une table. SVM cherche la meilleure ligne pour les séparer en laissant le plus d'espace possible entre elles.

Si ce n'est pas possible sur une table (2D), SVM peut transformer le problème en 3D et utiliser un plan pour mieux séparer les fruits.

SVM

Avantages :

- Efficace pour des petits ensembles de données
- Bonne performance sur des problèmes de classification
- Fonctionne bien même avec des données non linéaires (grâce aux noyaux)

Inconvénients :

- Peut être lent avec de grandes bases de données
- Sensible aux choix du noyau et des paramètres

SVM

Traduction en Python

```
from sklearn.svm import SVC
linear_SVM = SVC(kernel='linear')
linear_SVM.fit(X_train_sc,y_train)
```

```
y_predictSVM_l = linear_SVM.predict(X_test_sc)
```

```
svm_acc=accuracy_score(y_test,y_predictSVM_l)
svm_prec=precision_score(y_test,y_predictSVM_l)
svm_rec=recall_score(y_test,y_predictSVM_l)
print(confusion_matrix(y_test,y_predictSVM_l))
print('Accuracy linear SVM {0:.3f}'.format(svm_acc))
print('Precision linear SVM {0:.3f}'.format(svm_prec))
print('Recall linear SVM {0:.3f}'.format(svm_rec))
```

```
print(classification_report(y_test,y_predictSVM_l))
```

SVM

Principe général

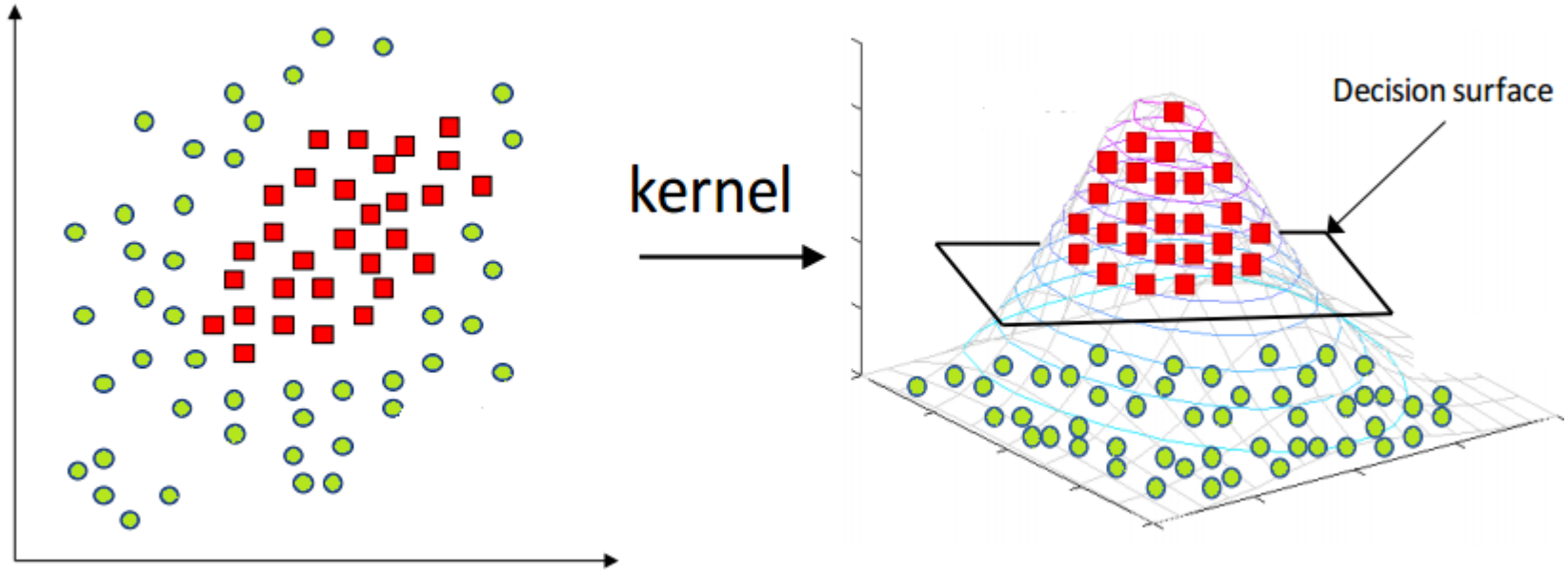
L'idée est de **transformer** les données dans un espace de plus haute dimension où elles deviennent séparables. Cette transformation est effectuée grâce à une **fonction noyau**.

Comment ça marche ?

- On applique une fonction noyau pour **projeter** les points de données dans un espace supérieur.
- Dans ce nouvel espace, on peut tracer une **ligne droite** pour séparer les classes.
- Puis on revient à l'espace original en utilisant les résultats obtenus.

SVM

Principe général



SVM

Principe général

Types de noyaux courants

- **Linéaire** : Pas de transformation (cas de base).
- **Polynomiale** : Transforme les données en utilisant des puissances.
- **RBF (Radial Basis Function, ou Gaussien)** : Très utilisé pour détecter des formes complexes.

SVM

Kernal SVM

Syntaxe

```
kernel_SVM = SVC(kernel='rbf')
# utilisation des données standardisées
kernel_SVM.fit(X_train_sc,y_train)
```

kernel='rbf' (noyau gaussien)

kernel='poly' (polynomial)

kernel='sigmoid'

```
y_predictSVM_k = kernel_SVM.predict(X_test_sc)
print(confusion_matrix(y_test,y_predictSVM_k))
print('Accuracy rbf SVM {0:.3f}'.format(accuracy_score(y_test,y_predictSVM_k)))
print('Precision rbf SVM {0:.3f}'.format(precision_score(y_test,y_predictSVM_k)))
print('Recall rbf SVM {0:.3f}'.format(recall_score(y_test,y_predictSVM_k)))
```

Logistic Regression

Principe

La régression logistique est un algorithme de machine learning supervisé utilisé pour les tâches de ***classification binaire*** (et parfois multi-classes). Contrairement à son nom, elle est davantage liée à la **classification** qu'à la régression.

La régression logistique modélise la probabilité qu'une observation appartienne à une catégorie donnée. Elle repose sur l'utilisation de la fonction sigmoïde pour transformer une combinaison linéaire des variables en une probabilité comprise entre 0 et 1.

Logistic Regression

Principe

Avantages

- Simple à implémenter et rapide.
- Interprétable (chaque coefficient représente l'effet d'une caractéristique sur la probabilité).
- Fonctionne bien avec des datasets de taille modérée.

Limites

- Ne modélise pas de relations complexes entre les variables (car basée sur une combinaison linéaire).
- Sensible aux valeurs aberrantes.
- Suppose une séparation linéaire des données (dans l'espace des caractéristiques).

Logistic Regression

Traduction en Python

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(X_train_sc,y_train)
```

```
y_predictLR = LR.predict(X_test_sc)
```

```
LR_acc=accuracy_score(y_test,y_predictLR)
LR_prec=metrics.precision_score(y_test,y_predictLR)
LR_rec=metrics.recall_score(y_test,y_predictLR)
print(confusion_matrix(y_test,y_predictLR))
print('Accuracy Logistic Regression {0:.3f}'.format(LR_acc))
print('Precision Logistic Regression {0:.3f}'.format(LR_prec))
print('Recall Logistic Regression {0:.3f}'.format(LR_rec))
```