



Epreuve d'informatique

1^{er} semestre AU : 2024 – 2025

Date : décembre 2024

Durée : 2H

Nombre de page : 6

Machine Learning : Apprentissage Supervisé

L'apprentissage supervisé est l'une des méthodes les plus populaires et largement utilisées en apprentissage automatique (machine learning). Il s'agit d'un type d'apprentissage où un modèle est formé à partir de données étiquetées, c'est-à-dire des données pour lesquelles les résultats (ou étiquettes) sont déjà connus.

Principe :

Dans l'apprentissage supervisé, le processus de formation consiste à utiliser un ensemble de données d'entrée (caractéristiques ou features) et leurs étiquettes correspondantes (classes ou target) pour entraîner un modèle. L'objectif est d'apprendre une fonction qui peut prédire les étiquettes ou résultats pour de nouvelles données non étiquetées.

Exemple d'Apprentissage Supervisé

Prenons l'exemple d'un jeu de données sur des fleurs d'Iris. Les caractéristiques des fleurs sont mesurées (longueur et largeur des sépales et des pétales) et étiquetées avec la target de la fleur (par exemple, Setosa, Versicolor, Virginica). L'objectif est de construire un modèle qui, étant donné les caractéristiques d'une nouvelle fleur, puisse prédire son étiquette.

Sepal Length	Sepal Width	Petal Length	Petal Width	Target (Classe)
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
6.2	2.8	4.5	1.5	Versicolor
7.1	3.0	5.9	2.1	Virginica

- Chaque **ligne** dans ce tableau est une instance (fleur).
- Les **colonnes** sont les caractéristiques de chaque instance
- La colonne "**Target**" représente l'étiquette associée à chaque instance

Étapes de l'Apprentissage Supervisé

1. Collecte des données :

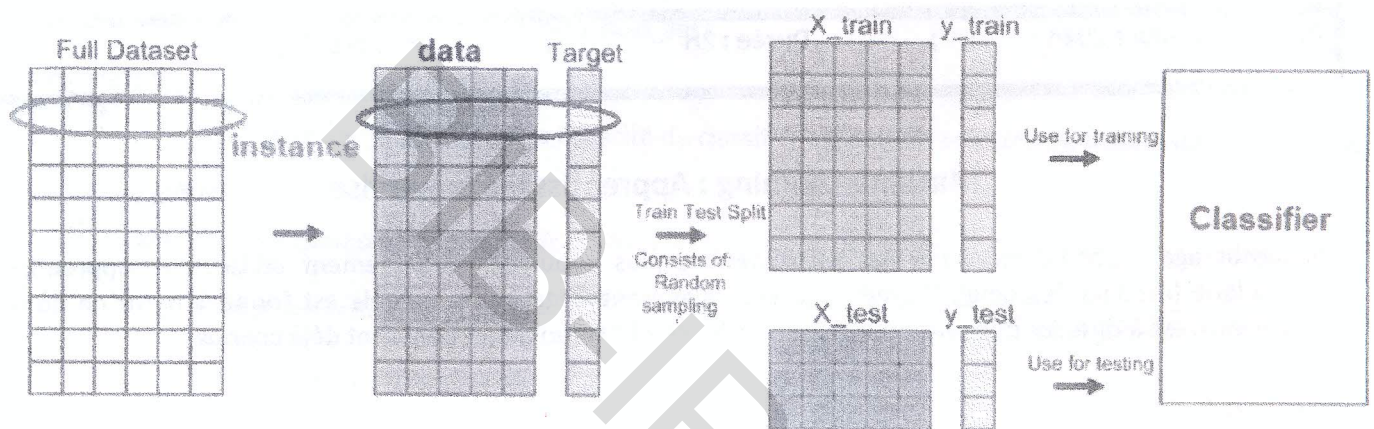
- Un jeu de données est constitué d'exemples d'entrée et de leurs étiquettes correspondantes (data et target). Ce jeu de données est souvent divisé en deux ensembles : un ensemble d'entraînement (X_train et y_train) et un ensemble de test (X_test et y_test).

2. Entraînement du modèle :

- Un algorithme d'apprentissage (comme la régression linéaire, les machines à vecteurs de support (SVM), les arbres de décision, ou KNN) est utilisé pour apprendre la relation entre les caractéristiques d'entrée et les étiquettes de sortie.

3. Évaluation du modèle :

- Une fois le modèle entraîné, il est testé sur un ensemble de données séparé (ensemble de test : X_test et y_test) pour évaluer sa performance. Les métriques d'évaluation courantes incluent la précision et le rappel.



Classe Instance

Question 1 :

Ajouter la méthode `__repr__(self)` qui renvoie une chaîne sous la forme :

'Features : [x₀, x₁, ..., x_{n-1}] : target = y'

Exemple :

```
>>>I1 = Instance([5.1, 3.5, 1.4, 0.2] , 'Setosa' )
>>>I2 = Instance([7.1, 3.0, 5.9, 2.1] , 'Virginica' )
>>>print("I1 = " , I1)
>>>print("I2 = " , I2)

I1 = Features : [5.1, 3.5, 1.4, 0.2] : target = Setosa
I2 = Features : [7.1, 3.0, 5.9, 2.1] : target = Virginica
```

Question 2 :

Ajouter la méthode `distance(self, other)` qui renvoie la distance euclidienne entre deux instances

Exemple :

```
>>> d = I1.distance(I2) #  $\sqrt{(5.1 - 7.1)^2 + (3.5 - 3.0)^2 + (1.4 - 5.9)^2 + (0.2 - 2.1)^2}$ 
>>> print("distance = ", d)
distance = 5.301886456724625
```

```
class Instance :
```

```
    def __init__(self , X , y=None) :
```

```
        self.X = X # une liste de réels (valeurs des Features)
```

```
        self.y = y # une chaîne (valeur du target)
```

classe Dataset

Attributs :

- **data** : une liste d'objets de la classe Instance
- **targets** : un ensemble de target(attribut y des objets Instance de la liste data)

Question 3 :

Ecrire la classe Dataset avec un constructeur `__init__(self, data)` prenant en paramètres une liste d'objets de la classe Instance.

Question 4 :

Ajouter la méthode `__getitem__(self, i)` qui renvoie la liste des valeurs des *Features* d'indice i (x_i) des objets Instance dans l'attributs data.

Exemple :

```
>>> data_iris = Dataset([I1 , I2])
>>> data_iris.__getitem__(0)
[5.1, 7.1]
```

Question 5 :

Ajouter la méthode `split_data(self, learning_percent = 80)` qui divise l'attribut data aléatoirement en deux objets DataSet (data_train et data_test) afin d'entraîner et de tester un modèle. Le nombre d'instance dans data_train égale à `learning_percent` % du nombre d'instance dans data (par défaut 80%).

Indication : utiliser la fonction `sample(L, k)` du module random qui retourne une liste contenant k éléments aléatoires uniques extraits d'une liste L.

Question 6 :

Ajouter la méthode `normalize(self)` qui transforme les attributs X des objets Instance de l'attribut data pour qu'elles se situent dans une plage spécifique selon la formule suivante :

$$x_{nor} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- x : la valeur d'origine
- x_{min} : la valeur minimale de la caractéristique (Feature)
- x_{max} : la valeur maximale de la caractéristique(Feature)

Cette méthode renvoie un nouvel objet Dataset avec un attribut data en tant qu'une liste d'objet Instance dont les valeurs de l'attribut X sont normalisées et l'attribut y garde celui de l'instance d'origine.

classe Classifier

Attributs :

- o **data_train** : un objet DataSet pour les données d'entraînement
- o **data_test** : un objet DataSet pour les données de test
- o **data_pred** : une liste d'objets Instance. C'est une copie de l'attribut *data* de l'objet *data_test* tel que l'attribut y de chaque objet Instance est initialisé à None.
- o **learning_percent** : pourcentage de division de la base d'entraînement

Question 7 :

Ecrire la classe Classifier avec un constructeur `__init__(self, objDataSet, learning_percent = 80)` qui :

- Divise l'objet Dataset `objDataSet` en deux objetDataSet (`data_train` et `data_test`) à l'aide de la méthode `split_data` de la classe DataSet.
- Initialise l'attribut `data_pred`
- Initialise l'attribut `learning_percent` avec le paramètre `learning_percent` par défaut 80%.

Question 8 :

Ajouter la méthode `accuracy(self)` qui mesure la performance d'un modèle de classification. Elle correspond au pourcentage de prédictions correctes parmi l'ensemble des prédictions.

Formule :
$$Acuuracy = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$

Explication : une prédiction est correcte si la target `y` dans une instance dans `data_pred` est la même target `y` dans la même instance dans `data_test`.

Question 9 :

Ajouter la méthode `rappel(self)` qui mesure la capacité du classifieur à identifier correctement les instances positives pour chaque target.

Exemple : Par exemple la dataSet iris contient 3 targets : `setosa`, `versicolor`, `virginica`

On calcule le $Rappel_{setosa} = \frac{\text{Nombre de prédictions correctes de la target setosa}}{\text{Nombre de prédictions correctes et incorrectes de la target setosa}}$

et de la même façon pour $Rappel_{versicolor}$ et $Rappel_{virginica}$

Cette méthode renvoie un dictionnaire dont chaque clé est la target de l'attribut `data_test` et la valeur correspondante est le rappel associé à chaque clé target.

Classe KNN : une sous classe qui hérite de la classe **Classifier**

Principe : Le classifieur **K-Nearest Neighbors (KNN)** est un algorithme d'apprentissage supervisé simple et efficace, utilisé pour les tâches de classification. Il repose sur la similarité entre les données, mesurée à l'aide de distances.

Pour prédire la target d'une nouvelle instance, KNN recherche les `k` instances les plus proches dans l'ensemble d'entraînement (`data_train`).

Attributs :

- o `K` : nombre d'instances les plus proches

Question 10 :

Ecrire la classe KNN avec un constructeur `__init__(self, objDataSet, learning_percent = 80, k = 3)`

Question 11 :

Ajouter la méthode ***predict_single(self, obj_instance)*** qui prend en paramètre un objet Instance pour prédire et affecter son target **y**. en suivant les étapes suivantes :

1. Extraire une liste triée des instances dans l'attribut `data_train` selon la distance entre le paramètre `obj_instance` et chaque instance de la `dataSet` `data_train`.
2. A partir de cette liste extraire la liste des `k` premiers instances (les plus proches)
3. Récupérer la liste des target (attribut `y`) de ces `k` plus proches instances
4. Remplir un dictionnaire dont les clés sont les target de `data_train` et les valeurs sont le nombre d'occurrence du target figuré dans les `k` plus proches instances.

Exemple de dictionnaire pour `k = 5` : `{ 'setosa' : 3, 'versicolor' : 0, 'virginica' : 2 }`

5. Chercher la target ayant un nombre d'occurrence maximum et l'affecter à l'attribut `y` de l'objet `obj_instance`

Question 12 :

Ajouter la méthode ***predict_all_test(self)*** qui applique la méthode `predict_single` pour toutes les instances de la liste `data_pred`

Programmation procédurale :

Question 13 :

Ecrire une fonction ***load_dataSet(nomF, sep)*** qui lit les lignes du fichier nommé `nomF` où chaque ligne contient les données d'une instance (`X` et `y`) séparé par `sep`. Cette méthode renvoie un objet `DataSet`.

```
Iris_data.csv
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
6.3,2.7,4.9,1.8,Iris-virginica
6.4,3.2,4.5,1.5,Iris-versicolor
4.7,3.2,1.3,0.2,Iris-setosa
7.7,2.8,6.7,2.0,Iris-virginica
```

Question 14 :

Ecrire la fonction ***head(dataSet, N)*** qui affiche les targets de la `dataSet` et ses `N` premiers instances

Exemple :

```
>>> dataSet_iris = load_dataSet("Iris_data.csv" , ',' )
>>> head(dataSet_iris , 5)
targets      : {'Iris-setosa', 'Iris-virginica', 'Iris-versicolor'}
Instance     : Features : [5.1, 3.5, 1.4, 0.2] : target = Iris-setosa
Instance     : Features : [4.9, 3.0, 1.4, 0.2] : target = Iris-setosa
Instance     : Features : [6.3, 2.7, 4.9, 1.8] : target = Iris-virginica
Instance     : Features : [6.4, 3.2, 4.5, 1.5] : target = Iris-versicolor
Instance     : Features : [4.7, 3.2, 1.3, 0.2] : target = Iris-setosa

>>> KnnCls = KNN(dataSet, 80 , 3)
>>> I = KnnCls.data_pred[0]
>>> print("instance de test      : " , I)
>>> KnnCls.predict_single(I)
>>> print("prediction du target : " , I)
instance de test      : Features : [5.1, 3.5, 1.4, 0.2] : target = None
prediction du target   : Features : [5.1, 3.5, 1.4, 0.2] : target = Iris-setosa

>>> KnnCls.predict_all_test()

>>>KnnCls.accuracy()
0.8666666666666667
>>> KnnCls.rappel()

{'Iris-setosa': 1.0, 'Iris-virginica': 0.7, 'Iris-versicolor': 0.9090909091}
```