UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ

Institut de la Francophonie pour l'Innovation

Apprentissage automatique

Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : 22

TP1 : KNN et arbre de décision TP2 : Perceptron

Rédigé par :

KINDA Zakaria

ZONGO Sylvain

KADIMA Olivier Kalombo

OUBDA Raphael Nicolas Wendyam

Enseignant :

Dr VU Thi Huong Giang

Année académique : 2018 - 2019

Table des matières

Table des figures

1 Introduction

En ce qui concerne l'apprentissage automatique, plusieurs définitions existent. Cependant nous allons le définir en ces termes : "L'apprentissage est un ensemble de mécanismes menant à l'acquisition de savoir-faire, de savoirs ou de connaissances ". Le but de ce premier TP est de faire une classification manuelle et automatique en utilisant deux algorithmes d'apprentissage supervisé notamment le plus proche voisin et l'arbre de décision. Pour la partie automatique, nous avons choisi d'utiliser le langage de programmation Python3. Dans ce présent rapport, nous allons présenter d'abord la méthode de K plus proches voisins, partie dans laquelle, nous allons classer manuellement de nouveaux individus, après nous allons présenter notre programme qui permet de faire un classement automatique. Ensuite, dans la seconde partie, nous parlerons de la méthode de l'arbre de décision notamment la construction de l'arbre, la définition des règles de déduction et le classement de nouveaux individus.

2 TP1 : K plus proche voisin (KNN) et 1rbre de décision

2.1 Classement des individus en utilisant 1NN et 3NN

2.1.1 Implémentation manuelle

Dans cette étape nous allons déterminer les individus les plus proches parmi d'autres individus. Avant d'effectuer ces calculs nous avons effectué les calculs des distances entre chaque individu à prédire et tous les individus de la base déjà classés. Nous allons donc utiliser la formule de de Manhatta pour le calcul de la distance. Cette distance utilise la formule ci-dessous avec valeur de q=1.

$$d(u,v)=\sqrt[q]{(|u_1-v_1|^q+|u_2-v_2|^q+...+|u_n-v_n|^q)}$$

FIGURE *1 – Formule de Manhatta*

Par suite, nous rangeons les valeurs obtenues par ordre croissant (du plus petit au plus grand) afin d'identifier les plus petites distances :

- Pour 1NN, nous retenons la classe du premier élément le plus proche ;
- Pour 3NN, nous retenons la classe majoritaire parmi les trois premiers éléments.

X1	X2	Classe 1NN	Classe 3NN
0,55	0,364	1	1

Distance L1	X1	X2	Class
0,058	0,606	0,366	1
0,092	0,488	0,334	1
0,106	0,478	0,398	1
0,12	0,542	0,252	1
0,192	0,506	0,512	0
0,192	0,428	0,294	1
0,298	0,376	0,488	0
0,392	0,394	0,6	0
0,418	0,312	0,544	0
0,512	0,298	0,624	0

FIGURE *2 – Résultats 1NN et 3NN pour la classe 1*

X1	X2	Classe 1NN	Classe 3NN
0,558	0,47	0	1

Distance L1	X1	X2	Class
0,094	0,506	0,512	0
0,152	0,606	0,366	1
0,152	0,478	0,398	1
0,2	0,376	0,488	0
0,206	0,488	0,334	1
0,234	0,542	0,252	1
0,294	0,394	0,6	0
0,306	0,428	0,294	1
0,32	0,312	0,544	0
0,414	0,298	0,624	0

FIGURE *3 – Résultats 1NN et 3NN pour la classe 2*

X1	X2	Classe 1NN	Classe 3NN
0,456	0,45	1	0

Distance L1	X1	X2	Class
0,074	0,478	0,398	1
0,112	0,506	0,512	0
0,118	0,376	0,488	0
0,148	0,488	0,334	1
0,184	0,428	0,294	1
0,212	0,394	0,6	0
0,234	0,606	0,366	1
0,238	0,312	0,544	0
0,284	0,542	0,252	1
0,332	0,298	0,624	0

FIGURE *4 – Résultats 1NN et 3NN pour la classe 3*

X1	X2	Classe 1NN	Classe 3NN
0,45	0,57	0	0

Distance L1	X1	X2	Class
0,086	0,394	0,6	0
0,114	0,506	0,512	0
0,156	0,376	0,488	0
0,164	0,312	0,544	0
0,2	0,478	0,398	1
0,206	0,298	0,624	0
0,274	0,488	0,334	1
0,298	0,428	0,294	1
0,36	0,606	0,366	1
0,41	0,542	0,252	1

FIGURE *5 – Résultat 1NN et 3NN pour la classe 4*

2.1.2 Tableau récapitulatif des résultats

Le tableau ci-dessous présente les résultats obtenu à travers le calcul des distances.

X1	X2	Classe 1NN	Classe 3NN
0,55	0,364	1	1
0,558	0,47	0	1
0,456	0,45	1	0
0,45	0,57	0	0

FIGURE 6 – *Tableau récapitulatif*

2.2 Implémentaion de l'algorithme KNN

2.2.1 Fonctionnement du programme

Le programme prend en entrée :

- Une base d'image d'apprentissage
- Une base de test
- le paramètre K (Les K éléments les plus proches)

En sortie le programmes donne :

- Le taux de prédiction de chaque classe
- Le nombre de K éléments plus proches

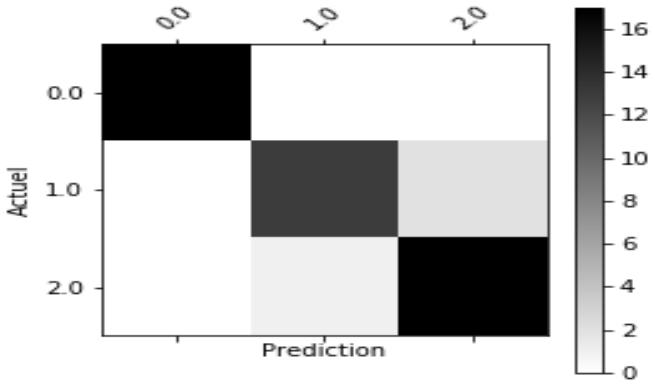
```
**********************************************
*                                            *
*            IMPLEMENTATION DE KNN           *
*                                            *
*                                            *
**********************************************

Entrer le lien de train ./iris/iris.trn

Entrer le lien de test ./iris/iris.tst

Entrer le K 3
Prediction  0.0  1.0  2.0
Actuel
0.0          17    0    0
1.0           0   15    0
2.0           0    1   17
```

Nous avons par la suite fait varier le paramètre K pour trouver la meilleure valeur de K qui donne la meilleure prédiction des classes.

2.2.2 Matrice de confusion

- Classification avec K=6

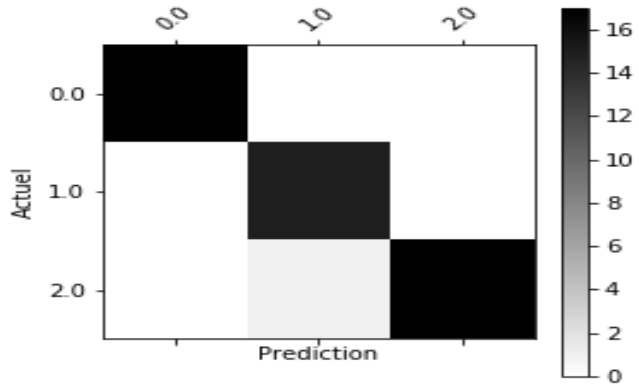
```
Entrer le K 6
Prediction  0.0  1.0  2.0
Actuel
0.0          17    0    0
1.0           0   13    2
2.0           0    1   17
```



Avec K=6 nous remarquons que 2 éléments ont été mal classés dans la classe 2.0 et pour bien illustrer cette erreur nous l'avons encadré en rouge.

- Classification avec K=3

```
Entrer le K 3
Prediction  0.0  1.0  2.0
Actuel
0.0          17    0    0
1.0           0   15    0
2.0           0    1   17
```



Avec K=3 nous remarquons que l'erreur précédente a été corrigée et pour bien illustrer cette correction nous l'avons encadrée en rouge. Les 2 éléments qui étaient mal classés dans la classe 2.0 sont maintenant bien classés.

2.3 Démonstration du théorème

Pour un suffisamment grand ensemble d'apprentissage de taille m, le taux d'erreur de 1NN est inférieure à deux fois le taux d'erreur minimal obtenu par la classification bayésienne. Soit R le taux d'erreur de 1NN et R* celui de Bayes. Par définition, on a :
$R = E[2(X)(1-(X))] = 2E[min(X), 1-(X)max(X), 1-(X)]$
en posant :
$min(X), 1-(X) = r*(X) et E(r*(X)) = R*$
on obtient les développements ci-après :
$R = 2E[r*(X)(1-r*(X))]$
$= 2E[r*(X) - r*2(X)]$
$= 2E[r*(X) - r*2(X) + R*2 - R*2 + 2R*r*(X) - 2R*r*(X)]$
$= 2E[r*(X) + R*2 - 2R*r*(x) - (r*(X) - R*)2$
$= 2E[r*(X) + R*2 - 2R*r*(x) - var(r*(X))]$
Or $2E[r*(X) + R*2 - 2R*r*(x) - var(r*(X))]2E[r*(X) + R*2 - 2R*r*(x)]$
$2(E[r*(X)] + E[R*2] - 2E[R*r*(x)]) = 2(R* + R*2 - 2R*2)$

Parconsquent R= 2 R*(1 - R*)

2.4 Arbre de décision

2.4.1 Construction de l'arbre de décision

La construction de l'arbre de décision va se faire à partir d'un ensemble d'apprentissage qui permettra de prédire la pratique du golf. La base de données est constituée de 5 variables et de 14 instances. Le tableau ci-dessous présente la base de données utilisée.

Outlook	Temperature	Humidity	Windy	Class
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
overcast	83	78	false	Play
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
overcast	64	65	true	Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
rain	75	80	false	Play
sunny	75	70	true	Play
overcast	72	90	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play

FIGURE 7 – *Base d'apprentissage*

2.4.2 Calcul des entropies

Nous allons calculer d'abord les entropies et les gains et choisissons l'attribut qui aura le gain le plus élevé comme racine de l'arbre de décision. Le calcul de l'entropie s'effectue en appliquant la formule ci-dessous.

$$H = -\sum_i p_i(\log_2 p_i)$$

FIGURE 8 – *Formule de l'entropie*

2.4.3 Les entropies après la première itération

Le calcul de la première itération nous permettra de déterminer la racine de notre arbre de décision

- Entropie de Outlook

	play	dontPlay	Nombre	Infos
Outlook/Sunny	2	3	5	0,970950594
Outlook/Rain	3	2	5	0,970950594
Outlook/Overcast	4	0	4	0
			H(Outlook)	0,693536139
			gain(Ouotlook)	0,24674982

FIGURE 9 – *Entropie outlook*

Pour le calcul de l'entropie de outlook, nous obtenons comme résultat de l'opération H(outlook)=0,6935

- Entropie Température

Pour le calcul de l'entropie de la température, nous faisons un trie croissant sur la base, afin de classer les valeurs de la température par ordre croissant, en fonction de la classe, pour faciliter le découpage par intervalle, donc le calcul de l'entropie. Lorsque nous faisons le trie nous obtenons le tableau ci-dessous

Outlook		Temperature	Humidity	Windy	Class
overcast		64	65	true	Play
rain		65	70	true	Don't Play
rain		68	80	false	Play
sunny		69	70	false	Play
rain		70	96	false	Play
rain		71	80	true	Don't Play
sunny		72	95	false	Don't Play
overcast		72	90	true	Play
rain		75	80	false	Play
sunny		75	70	true	Play
sunny		80	90	true	Don't Play
overcast		81	75	false	Play
overcast		83	78	false	Play
sunny		85	85	false	Don't Play

FIGURE *10 – Tableau pour le calcul de l'entropie température*

Après le calcul des entropies de la température en fonction des intervalles, nous prenons la plus petite entropie. Nous obtenons donc le résultat suivant.

	play	dontplay	Nombre	infos
Temperature/=<83	9	4	13	0,89049164
Temperature/>83	0	1	1	0
			Entropie Moyenne	0,826885094
			gain(Temperature)	0,113400864

FIGURE *11 – Entropie de la température*

- Entropie de l'humidité

Dans le calcul de l'entropie de l'humidité, nous effectuons la même procédure que celle de la température, nous obtenons alors le résultat suivant.

	play	dontplay	Nombre	infos
Humidity/=<65	1	0	1	0
Humidity/>65	8	5	13	0,961236605
			Entropie moyenne	0,892576847
			gain(Humidity)	0,047709111

FIGURE *12 – Entropie de l'humidité*

- Entropie Windy

Pour le calcul de l'entropie de windy, nous effectuons le trie sur la variable et nous calculons

l'entropie en découpant les différentes instance de la variable. Nous obtenons ainsi le résultat ci-dessous.

	play	dontplay	Nombre	infos
Windy/true	3	3	6	1
Windy/false	6	2	8	0,811278124
			Entropie Moyenne	0,892158928
			gain(Windy)	0,04812703

FIGURE *13 – Entropie de windy*

Après le calcul des entropies de la première itération nous obtenons outlook comme la racine de notre arbre de décision comme le présente la figure ci-dessous.

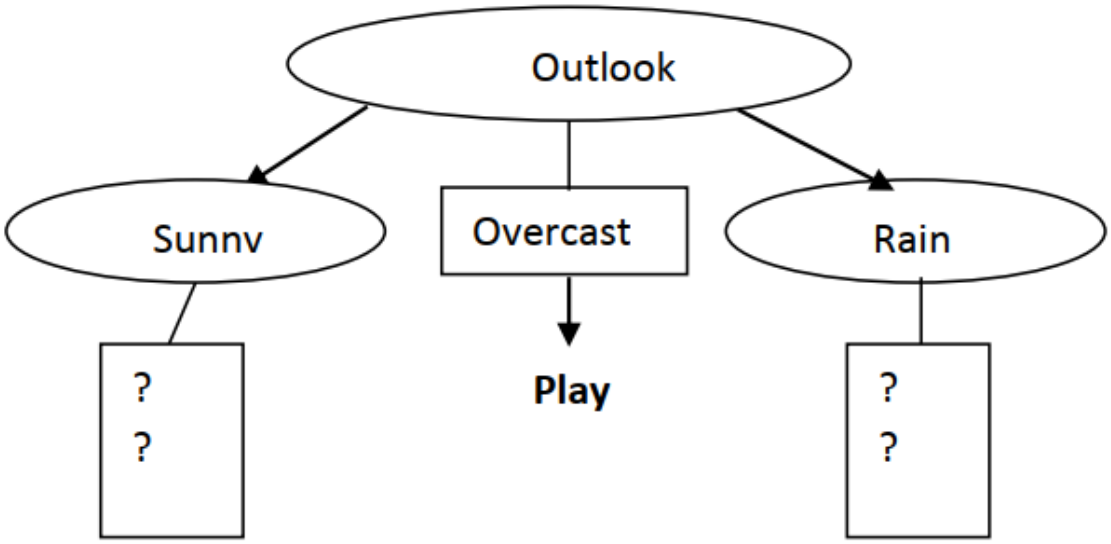


FIGURE *14 – Racine de l'arbre de décision*

2.4.4 Les entropies après la deuxième itération

Dans le calcul de la première itération, nous avons déterminé la racine de l'arbre de décision pour la suite de la construction de l'arbre, nous allons calculer les entropies des variables température, humidité et windy. A cet effet dans cette deuxième itération et calcul des entropies en ignorant l'attribut Outlook qui a été choisi comme RACINE DE L'ARBRE après. Nous allons donc Commencer par ordonner les individus de l'attribut Outlook et considérons seulement ceux qui ne sont pas directement reliés à une classe(entropie $=!$ 0).

Outlook	Temperature	Humidity	Windy	Class
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
overcast	83	78	false	Play
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
overcast	64	65	true	Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
rain	75	80	false	Play
sunny	75	70	true	Play
overcast	72	90	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play

FIGURE *15 – Données de la deuxième itération*

- Entropie de Windy

	Extraction des indivus rain et tri par Windy			
	Temperature	Humidity	Windy	Class
rain	70	96	false	Play
rain	68	80	false	Play
rain	75	80	false	Play
rain	65	70	true	Don't Play
rain	71	80	true	Don't Play

Calcul de l'entropie Windy

	play	dontplay	nombre	infos
Windy/false	3	0	3	0
Windy/true	0	2	2	0
			Entropie moyenne	0

FIGURE *16 – Entropie windy*

- Entropie de la température

Extraction des indivus et tri par temperature

	Temperature	Humidity	Windy	Class
sunny	69	70	false	Play
sunny	72	95	false	Don't Play
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play

	play	dontplay	nombre	infos
temp/<=75	2	1	3	0,918295834
temp/<=75	0	2	2	0
			Entropie moyenne	0,5509775

FIGURE *17 – Entropie température*

- Entropie de humidité

Extraction des individus et tri par Humidity

	Temperature	Humidity	Windy	Class
sunny	69	70	false	Play
sunny	75	70	true	Play
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
sunny	72	95	false	Don't Play

Calcul de l'entropie de Humidity

	play	dontplay	nombre	infos
Humidity/<=70	2	0	2	0
Humidity/>70	0	3	3	0
			Entropie moyenne	0

FIGURE *18 – Entropie humidité*

Après calcul de l'entropie de Température, Humidity et Windy en éliminant l'attribut Outlook, nous obtenons :
H(Windy)=0 suivant (rain)
H(Humidy)=0 suivant (sunny)
H(Temperature)=0,55
Nous pouvons conclure que la température n'a pas assez d'influence sur la prédiction de la pratique de Golf par conséquent l'attribut Température peut être ignoré.

2.4.5 Construction de l'arbre de décision

La construction de l'arbre de décision se fait en utilisant les entropies ou les gains. En utilisant l'entropie, nous construction l'arbre en tenant compte de la valeur minimale des entropies. Cependant en utilisant les gains nous choisissons la valeur maximale des différents gains. Nous obtenons alors l'arbre de décision ci-dessous.

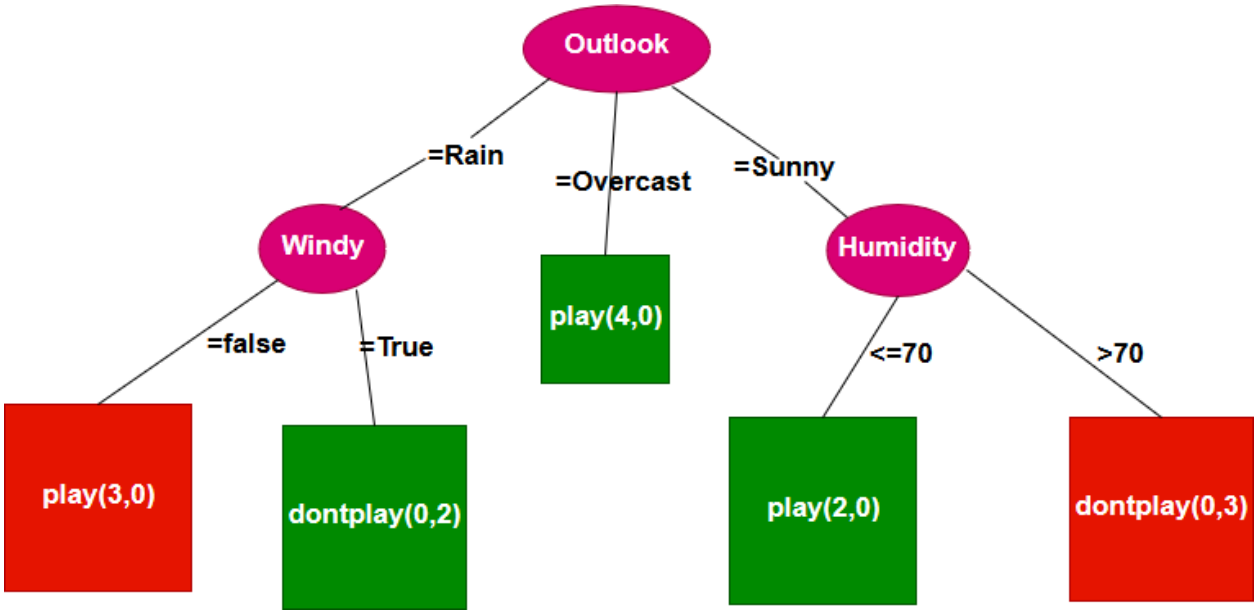


FIGURE *19 – Arbre de décision*

De cet arbre nous pouvons extraire les règles de décision suivante :

- Si (Outlook=Overcast) alors Golf = play ;
- Si (Outlook=Rain) et (Windy=false) alors Golf = play ;
- Si (Outlook=Rain) et (Windy=true) alors Golf = dontplay ;
- Si (Outlook=Sunny) et (Humidity<=70) alors Golf = play ;
- Si (Outlook=Sunny) et (Humidity > 70) alors Golf= dontplay.

2.4.6 Classification de ces individus grâce aux règles extraites

Pour la classification des individus, nous allons utiliser les règles extraites de l'arbre de decision ci-dessous. Nous obtenons alors le résultat suivant.

Outlook	Temperature	Humidity	Windy	Class	Règle
overcast	63	70	false	play	a
rain	73	90	true	play	c
sunny	70	73	true	dontplay	e

FIGURE *20 – Tableau de classification des individus*

2.5 Expliquer pourquoi l'ensemble d'arbres de décision améliore la prédiction d'un seul

Lorsque nous avons un seul arbre la variance est plus élevée que si nous avons plusieurs arbres la variance de l'ensemble est obtenue en faisant la moyenne des variances de tous les arbres ce qui réduire la variance permet d'améliorer la prédiction d'un seul. Nous pouvons illustrer cette amélioration avec le diagramme suivant :

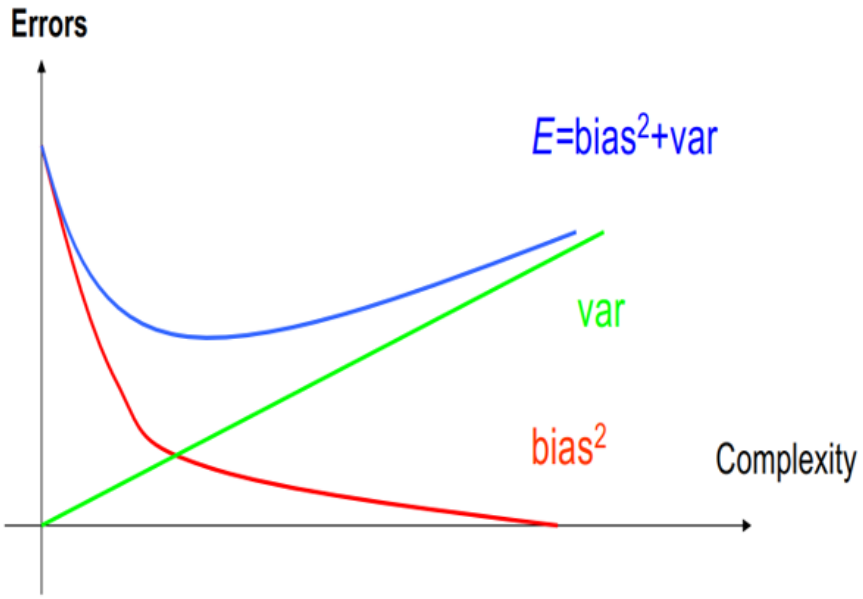


FIGURE *21 – Schéma d'explication pour l'amélioration de la prédiction*

2.6 Démontrer que le Bagging de k-plus proche voisin n'améliore pas le comportement d'un seul

Le bagging est une méthode ensembliste, illustrons le fonctionnement de cette méthode par le diagramme suivant :

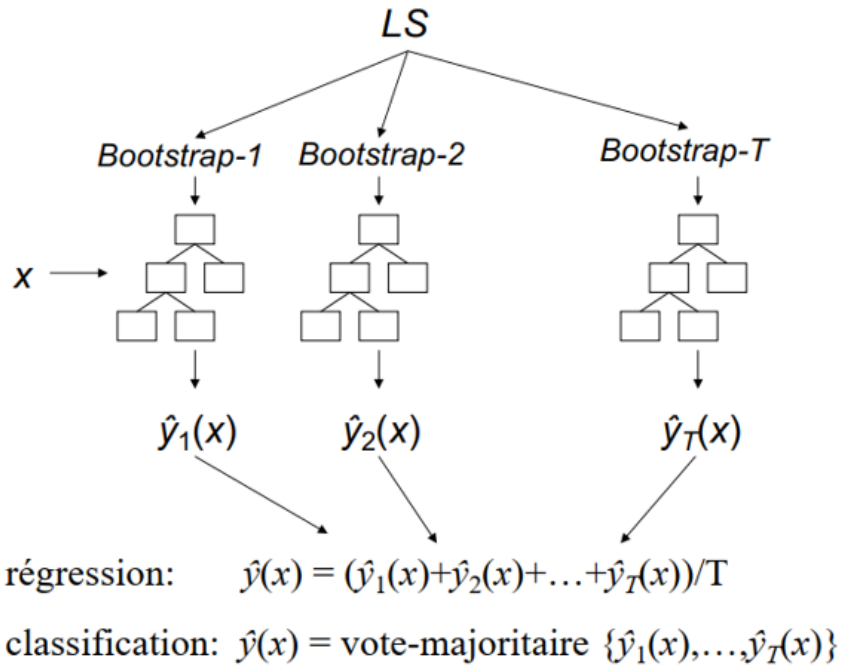


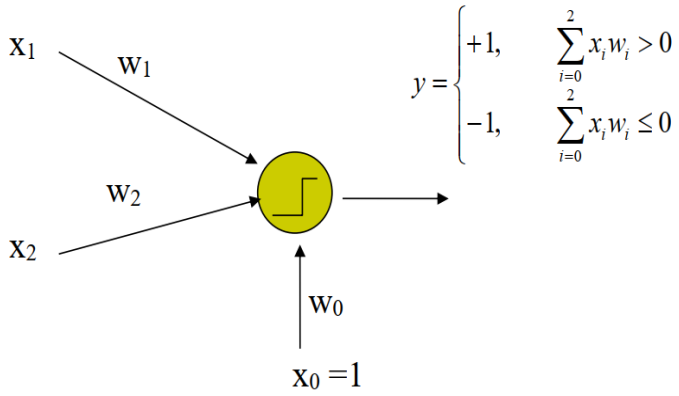
FIGURE *22 – Schéma d'illustration de Baggin*

Il consiste à sous-échantillonner le training set et de faire générer à l'algorithme voulu un modèle pour chaque sous-échantillon. On obtient ainsi un ensemble de modèles dont il convient de faire la moyenne (lorsqu'il s'agit d'une régression) ou de faire voter (pour une classification) les différentes prédictions. L'utilisation du Bagging est adaptée aux algorithmes à forte variance, complexes ou instables pour améliorer leur performance (réseaux neuronaux, arbres de décision pour la classification ou la régression...), mais il peut également dégrader les qualités pour des algorithmes plus stables (k plus proches voisins avec k grand, régression linéaire. Aussi le résultat obtenu avec le KNN n'est reste la même avec le bagging, donc nous pouvons conclure qu'un bagging du KNN détériorera la stabilité du KNN.

3 TP2 : Perceptron

L'objectif de ce TP est d'implémenter dans un premier temps un reseau de neurone avec une seule couche(percetron) et dans un second temps avec multi-couches à l'aide de TensorFlow et Keras.

3.1 Exercice 1



A partir cette équation $W1.X1 + W2.X2 + b = 0$ obtenue à partir du schéma ci-dessus nous allons suivre les étapes suivantes pour l'implémentation de perceptron.

- Construction du graphe
- Implémentation de la fonction de perte

La formule ci-dessous est la fonction de perte.

$$loss = \sum_{i=1}^{m} \left(y^{(i)} - predict^{(i)} \right)^2$$

FIGURE *23 – fonction de perte*

- Entrainer le perceptron avec notre jeune de données
- Enfin prédire la sortie du modèle

Comme indiqué sur le schéma ci-dessous le programme prend en entrée :

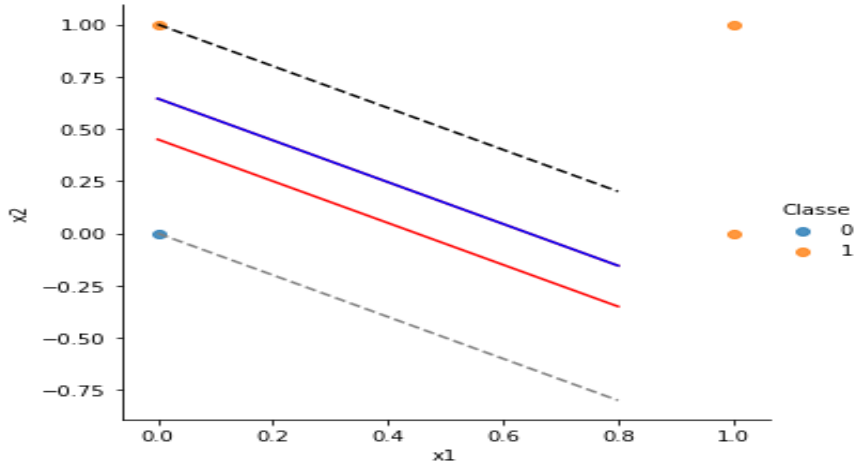
- les entrées x1 et x2
- les poids w1 et w2
- b

Ainsi avec ces entrées il pourra prédire la sortie y(i) qui est une classe (0 ou 1) comme suit :

```
prediction =  [[0]
 [1]
 [1]
 [1]]
W1 =  5.1203914
W2 =  5.1205163
b =  -2.3015034
```

Comme demandé dans l'énoncé nous avons construit une droite avec le perceptron et une autre avec le SVM représenté par la figure ci-dessous :

```
prediction =  [[0]
 [1]
 [1]
 [1]]
W1 =  5.1203914
W2 =  5.1205163
b =  -2.3015034
```



- Droite Blue : Séparation avec Perceptron
  Le perceptron cherche uniquement une droite permettant de séparer les deux classes sans aucune contrainte

- Droite Rouge : Séparation avec SVM
  SVM sépare au mieux les deux classes car il cherche la droite optimale entre les deux classes

3.2 Exercice 2

Les entrées du programmes sont :

- nom fichier de la base d'apprentissage
- nom fichier de la base de test
- h (le pas d'apprentissage)

- maxit (le nombre d'itérations)

Nous avons ici utilisé la base de données SPAM, nous avons d'abord divisé la base de données en données d'apprentissage et test.

```
========>   EXERCICE 2 AVEC LE DATASET SPAM   <=======

Entrer le nombre d'itération:23
prediction =  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]

 [-1.53076706e+01]
 [-8.02687912e+01]
 [-1.13070946e+02]
 [-8.43851395e+01]
 [-1.11484432e+00]
 [-1.07625360e+01]
 [-1.07025051e+01]
 [-5.60209312e+01]
 [-4.34789276e+00]
 [-1.16831390e+02]
 [-3.01222305e+01]
 [-6.80912638e+00]
 [-1.48345874e+03]
 [-1.84721152e+04]
 [-1.31568781e+05]]
b =  -428.2864
```

Nous obtenons des résultats peu satisfaisant car les données ne sont pas linéaires et nous avons utilisé un perceptron simple ce qui explique bien le fait que les prédictions sont mauvaises .

3.3 Exercice 4 : Impléméntation des reseaux de neurones avec Keras

Entrainons le réseau et calculons l'exactitude (accuracy) du modèle sur l'ensemble d'apprentissage.
Nous avos fait varier le nombre d'itérations et le pas d'apprentissage afin de trouver ceux qui permet de donner les meilleures prédictions.

3.3.1 Nombre d'itérations=1000 et pas d'apprentissage = 0.5

```
=======>       APPRENTISSAGE AVEC KERAS       <=======


Entrer le nombre d'itération: 1000

Entrer le pas: 0.5
Epoch 1/1000
4/4 [==============================] - 2s 427ms/step - loss: 0.6450 - acc: 0.7500
Epoch 2/1000
4/4 [==============================] - 0s 860us/step - loss: 0.6417 - acc: 1.0000
Epoch 3/1000
4/4 [==============================] - 0s 894us/step - loss: 0.6384 - acc: 1.0000
Epoch 4/1000
4/4 [==============================] - 0s 993us/step - loss: 0.6351 - acc: 1.0000
Epoch 5/1000
4/4 [==============================] - 0s 651us/step - loss: 0.6319 - acc: 1.0000
Epoch 6/1000
4/4 [==============================] - 0s 1ms/step - loss: 0.6287 - acc: 1.0000
Epoch 7/1000
4/4 [==============================] - 0s 667us/step - loss: 0.6255 - acc: 1.0000
Epoch 8/1000

.
.
.
4/4 [==============================] - 0s 683us/step - loss: 0.0843 - acc: 1.0000
Epoch 995/1000
4/4 [==============================] - 0s 763us/step - loss: 0.0843 - acc: 1.0000
Epoch 996/1000
4/4 [==============================] - 0s 686us/step - loss: 0.0842 - acc: 1.0000
Epoch 997/1000
4/4 [==============================] - 0s 1ms/step - loss: 0.0841 - acc: 1.0000
Epoch 998/1000
4/4 [==============================] - 0s 926us/step - loss: 0.0840 - acc: 1.0000
Epoch 999/1000
4/4 [==============================] - 0s 754us/step - loss: 0.0839 - acc: 1.0000
Epoch 1000/1000
4/4 [==============================] - 0s 927us/step - loss: 0.0839 - acc: 1.0000
[array([[-1.04339  ],
       [ 0.9918884]], dtype=float32), array([0.], dtype=float32)]
```

3.3.2 Nombre d'itérations=30 et pas d'apprentissage = 0.3

```
=======>       APPRENTISSAGE AVEC KERAS       <=======


Entrer le nombre d'itération: 30

Entrer le pas: 0.3
Epoch 1/30
4/4 [==============================] - 2s 482ms/step - loss: 0.4127 - acc: 0.7500
Epoch 2/30
4/4 [==============================] - 0s 716us/step - loss: 0.4113 - acc: 1.0000
Epoch 3/30
4/4 [==============================] - 0s 817us/step - loss: 0.4100 - acc: 1.0000
Epoch 4/30
```

```
Epoch 25/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3819 - acc: 1.0000
Epoch 26/30
4/4 [==============================] - 0s 648us/step - loss: 0.3807 - acc: 1.0000
Epoch 27/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3795 - acc: 1.0000
Epoch 28/30
4/4 [==============================] - 0s 661us/step - loss: 0.3784 - acc: 1.0000
Epoch 29/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3772 - acc: 1.0000
Epoch 30/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3760 - acc: 1.0000
[array([[ 0.16743255],
       [-0.01891959]], dtype=float32), array([0.], dtype=float32)]
```

3.3.3 Nombre d'itérations=10 et pas d'apprentissage = 0.5

```
=======>       APPRENTISSAGE AVEC KERAS       <========

Entrer le nombre d'itération: 10

Entrer le pas: 0.5
Epoch 1/10
4/4 [==============================] - 2s 407ms/step - loss: 0.6850 - acc: 0.5000
Epoch 2/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6814 - acc: 0.7500
Epoch 3/10
4/4 [==============================] - 0s 2ms/step - loss: 0.6779 - acc: 0.7500
Epoch 4/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6743 - acc: 0.7500
Epoch 5/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6708 - acc: 0.7500
Epoch 6/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6674 - acc: 0.7500
Epoch 7/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6639 - acc: 0.7500
Epoch 8/10
4/4 [==============================] - 0s 2ms/step - loss: 0.6605 - acc: 0.7500
Epoch 9/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6572 - acc: 0.7500
Epoch 10/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6538 - acc: 0.7500
[array([[ 0.67751586],
       [-1.3838701 ]], dtype=float32), array([0.], dtype=float32)]
```

3.3.4 Nombre d'itérations=100000 et pas d'apprentissage = 0.5

```
Epoch 99993/100000
4/4 [==============================] - 0s 914us/step - loss: 9.3998e-04 - acc: 1.0000
Epoch 99994/100000
4/4 [==============================] - 0s 881us/step - loss: 9.3998e-04 - acc: 1.0000
Epoch 99995/100000
4/4 [==============================] - 0s 722us/step - loss: 9.3992e-04 - acc: 1.0000
Epoch 99996/100000
4/4 [==============================] - 0s 918us/step - loss: 9.3992e-04 - acc: 1.0000
Epoch 99997/100000
4/4 [==============================] - 0s 1ms/step - loss: 9.3992e-04 - acc: 1.0000
Epoch 99998/100000
4/4 [==============================] - 0s 1ms/step - loss: 9.3992e-04 - acc: 1.0000
Epoch 99999/100000
4/4 [==============================] - 0s 861us/step - loss: 9.3992e-04 - acc: 1.0000
Epoch 100000/100000
4/4 [==============================] - 0s 2ms/step - loss: 9.3989e-04 - acc: 1.0000
[array([[ 0.08484232],
       [-1.2711526 ]], dtype=float32), array([0.], dtype=float32)]
```

3.4 Tableau récapitulatif

Nous avons regroupé la variation des différents paramètres dans un tableau afin de mieux comprendre l'influence de ces variables sur le taux d'erreur et l'accuracy

Nombre d'itérations	pas d'apprentissage	loss	accurracy
1000	0.5	0.0839	1.000
30	0.3	0.3760	1.000
10	0.5	0.6538	0.7500
100000	0.5	9.3989e-04	0.10000

TABLE *1 – Tableau comparatif*

Pour le même pas d'apprentissage nous remarquons que plus le nombre d'itérations augmente moins est le taux d'erreurs. C'est le cas des itérations 1000 , 10 et 100000 avec le même pas d'apprentissage 0.5.

4 Conclusion

Dans ce premier TP il s'agissait de mettre en œuvre l'apprentissage supervisé pour prédire la classification des individus avec les méthodes de k plus proches voisins et l'arbre de décision. A cet effet, nous avons conçu un programme en Python pour nous permettre de faire cette prédiction. Notre programme prend en entrée les bases d'apprentissage et de test, le nombre de voisins et la distance de calcul. En sortie. Il fournit la matrice de confusion et le taux de bon classement. Lors des expérimentations, nous avons constaté que le choix du k, nombre de voisins relève surtout d'un savoir-faire spécialisé. Aussi, la distance euclidienne offre en général de meilleurs taux que celle de Manhattan.

Le but de ce deuxième TP était de mettre en œuvre l'apprentissage supervisé pour prédire la classification des individus avec les méthodes de réseaux de neurones et les Séparateurs à Vaste Marge (SVM). Ce programme a été implémenté en utilisant le langage python. Nous avons dans un premier temps implémenté le perceptron simple puis le perceptron multicouche et l'implémentation avec keras. Notre programme prend en entrée les bases d'apprentissage et de test, le nombre de voisins et la distance de calcul et en sortie, il fournit la matrice de confusion et le taux de bon classement.

En somme, nous pouvons dire que ces travaux pratiques (1 et 2) ont été très bénéfique et nous ont permis d'avoir une meilleure compréhension de l'apprentissage automatique notamment les méthodes kNN, l'arbre de décision, les réseaux de neurones et les SVM.