

# Projet K-PPV

Reconnaissance de formes et analyse d'images

Réalisée par :  
EL MEHDI EL MAKOUL  
Ayoub EL ATTAR

**Master 1 BDMA**  
**Promotion 2023-2024**

## Table des matières

Première étape : .....	4
Question 2 - implémenter la méthode computeDistances: .....	4
Question 3 - implémenter la méthode findClass: .....	4
Question 4 - La procédure d'évaluation.....	5
Question 5 : Convertir 1-PPV en un véritable K-PPV .....	6
Question 6 - Cross Validation .....	8
Deuxième étape : WEKA.....	9
La découverte de Weka .....	9
Classification "IBK" .....	10
Neural networks .....	11
Réduction le nombre d'epochs à 25 .....	13
GUI option .....	14
Tests on MNIST dataset .....	15
L'algorithme IBK .....	15

# Table de Figures

Figure 1: La méthode ComputeDistances .....	4
Figure 2: La méthode findClass.....	5
Figure 3: la resultat de matrice de confusion et taux de connaissance.....	5
Figure 4: La méthode PredictClassKNearestNeighbours.....	6
Figure 5: La méthode crossValidation.....	8
Figure 6: la résultat données par la cross validation .....	9
Figure 7:découverte de Weka.....	10
Figure 8:Resulat avec la classification IBX(50% split).....	11
Figure 9:Resulat avec la classification Multilayer Perceptron (50% split).....	12
Figure 10:Le graphique de prédiction(500epochs) .....	13
Figure 11:Le graphique de prédiction(25 epochs).....	14
Figure 12:Le réseau.....	15

## Première étape :

Nous avons suivi les instructions fournies dans le premier TP en utilisant l'outil Eclipse pour créer un nouveau projet Java nommé kPPV, avec un package appeler KPPV. Ensuite, nous avons ajouté le fichier source kPPV.java dans le dossier src du projet, et copié le fichier "iris.data" dans le dossier racine du projet. Enfin, nous avons testé le code du programme en utilisant le débogueur intégré pour garantir son bon fonctionnement.

## Question 2 - implémenter la méthode computeDistances:

Nous avons implémenté la fonction **ComputeDistances** en utilisant la méthode de la distance Euclidienne pour calculer la distance entre un vecteur d'entrée et tous les exemples dans l'ensemble d'apprentissage. Pour ce faire, nous avons utilisé une double boucle for : la première parcourt tous les exemples d'apprentissage dans toutes les classes, tandis que la seconde parcourt toutes les caractéristiques de chaque exemple. À chaque itération, nous avons calculé la distance euclidienne entre le vecteur d'entrée x et l'exemple, et stocké cette distance dans la variable "distance". Enfin, nous avons ajouté cette distance au tableau "distances". Ainsi, à la fin de la fonction, le tableau "distances" est rempli avec toutes les distances calculées.

```
private static void ComputeDistances(Double x[], Double distances[]) {
    // Initilisation
    int index = 0;
    for (int c = 0; c < NbClasses; c++) {
        for (int n = 0; n < NbExLearning; n++) {
            Double[] example = data[c][n];
            Double distance = 0.0;
            for (int f = 0; f < NbFeatures; f++) {
                distance += Math.pow(x[f] - example[f], 2);
            }
            distances[index] = Math.sqrt(distance);
            index++;
        }
    }
}
```

Figure 1: La méthode ComputeDistances

## Question 3 - implémenter la méthode findClass:

Nous avons entrepris de créer une nouvelle fonction visant à déterminer la classe correspondante à un vecteur d'entrée en utilisant la table de distances, calculée dans la question 2. Pour ce faire, nous avons initié la création d'une fonction nommée **findClass**.

```
private static int findClass(Double distances[]) {
    // Initialisation de la distance minimale et de l'indice correspondant
    double minDistance = distances[0];
    int minIndex = 0;
    // Parcours des distances pour trouver la plus petite
    for (int i = 1; i < distances.length; i++) {
        if (distances[i] < minDistance) {
            minDistance = distances[i];
            minIndex = i;
        }
    }
    // Calcul de l'indice de classe en utilisant le nombre d'exemples par
    classe
    return minIndex / NbExLearning;
}
```

**Figure 2: La méthode findClass**

La fonction **findClass** prend en paramètre un tableau de distances entre le point d'entrée et les exemples d'apprentissage. Elle initialise une variable 'min' avec la première distance du tableau et un indice associé. Ensuite, elle parcourt le tableau de distances à partir du deuxième élément. À chaque itération, si la distance en cours est inférieure à 'minDistance', elle met à jour 'minDistance' avec cette nouvelle valeur et actualise l'indice correspondant. Sinon, elle passe à l'itération suivante. Finalement, la fonction retourne l'indice divisé par le nombre total d'exemples d'apprentissage pour obtenir la classe prédite du point d'entrée.

## Question 4 - La procédure d'évaluation

La procédure d'évaluation est la suivante : Pour chaque exemple de test, nous calculons la distance entre cet exemple de test et chaque exemple d'apprentissage de toutes les classes en utilisant la méthode computeDistances. Ensuite, en utilisant notre fonction findClass, nous déterminons la classe prédite pour l'exemple de test en utilisant l'algorithme 1-PPV.

Nous stockons ensuite cette classe prédite dans un tableau de prédictions et mettons à jour la matrice de confusion en fonction de la classe prédite et de la classe réelle de l'exemple de test.

Une fois cette procédure d'évaluation terminée pour tous les exemples de test, nous calculons le taux de reconnaissance et construisons la matrice de confusion. Ci-dessous,

```
Matrice de confusion :
25      0      0
0       24     1
0        5    20
Recognition rate: 0.92
```

**Figure 3: la resultat de matrice de confusion et taux de connaissance**

Nous présentons les résultats de la matrice de confusion et du taux de reconnaissance obtenus. Pour évaluer les performances de notre algorithme 1-PPV,

**Le taux de reconnaissance**, qui représente le pourcentage d'exemples de test correctement classés dans notre exemple on a trouvé 0.92 donc 92% des classes sont correctement classés,

**la matrice de confusion**, une matrice carrée indiquant le nombre d'exemples appartenant à chaque classe qui ont été prédits comme appartenant à chaque classe.

## Question 5 : Convertir 1-PPV en un véritable K-PPV

Pour adapter l'algorithme 1-PPV en K-PPV, nous avons commencé par introduire une nouvelle classe nommée **PredictClassKNearestNeighbours**. Cette classe est conçue pour tenir compte des K voisins les plus proches plutôt que d'un seul voisin comme dans le cas de 1-PPV.

Pour la variable **KNeighbors**, une portion de code assure que l'utilisateur fournit le nombre de voisins requis lorsqu'il exécute le programme via la ligne de commande, comme illustré ici avec K=5 : « java kppv.KPPV 5 ».

```
private static int PredictClassKNearestNeighbours(Double distances[], int
KNeighbors){
    // Création d'un tableau pour stocker le nombre d'occurrences de chaque
classe
    int[] classOccurrences = new int[NbClasses];
    Arrays.fill(classOccurrences, 0);

    // Tri des distances par ordre croissant et sélection des K plus proches
voisins
    Double[] sortedDistances = distances.clone();
    Arrays.sort(sortedDistances);
    for (int i = 0; i < KNeighbors; i++) {
        Double currentDistance = sortedDistances[i];
        for (int j = 0; j < distances.length; j++) {
            if (distances[j] == currentDistance) {
                int currentClass = j / NbExLearning;
                classOccurrences[currentClass]++;
                break;
            }
        }
    }
    // Sélection de la classe majoritaire parmi les voisins
    int maxOccurrences = 0;
    int predictedClass = 0;
    for (int i = 0; i < NbClasses; i++) {
        if (classOccurrences[i] > maxOccurrences) {
            maxOccurrences = classOccurrences[i];
            predictedClass = i;
        }
    }
    return predictedClass;
}
```

Figure 4: La méthode PredictClassKNearestNeighbours

PredictClassKNearestNeighbours, vise à prédire la classe d'un point en utilisant l'algorithme K-PPV (K plus proches voisins). Tout d'abord, elle crée un tableau pour stocker le nombre d'occurrences de chaque classe, initialisant toutes les occurrences à zéro. Ensuite, elle trie les distances par ordre croissant et sélectionne les K plus proches voisins. Pour chaque distance

dans les K plus proches voisins, elle détermine la classe correspondante et met à jour le nombre d'occurrences de cette classe. Enfin, elle sélectionne la classe majoritaire parmi les voisins en comparant le nombre d'occurrences de chaque classe et retourne la classe prédite. En résumé, cette fonction utilise l'algorithme K-PPV pour prédire la classe d'un point en se basant sur les K voisins les plus proches.

Voici ci-dessous les résultats obtenus pour les différents k testés :

- Pour « java kppv.KPPV 5> → recognition Rate: 0.92
- Pour « java kppv.KPPV 10> → recognition Rate: 0.9333333333333333
- Pour « java kppv.KPPV 15> → recognition Rate: 0.9333333333333333
- Pour « java kppv.KPPV 20> → recognition Rate: 0.9466666666666667
- Pour « java kppv.KPPV 21> → recognition Rate: 0.9466666666666667
- Pour « java kppv.KPPV 30> → recognition Rate: 0.9066666666666666

Selon nos résultats, le taux de reconnaissance pour k=5 atteint la valeur de 0.92. Cependant, pour k=10 et k=20, le taux de reconnaissance augmente, avec des valeurs respectives de 0.933 et 0.947. En revanche, pour k=30, le taux de reconnaissance diminue légèrement, avec une valeur de 0.907. Ces observations suggèrent que la valeur optimale de k se situe probablement entre 15 et 21, car le taux de reconnaissance est le plus élevé dans cette plage.

## Question 6 - Cross Validation

```
private static double crossValidation(int k, int nFolds) {
    // Initialisation de l'accuracy totale
    double accuracyTotale = 0;
    int taillePli = NbExLearning / nFolds;
    // Parcours de tous les plis
    for (int fold = 0; fold < nFolds; fold++) {
        int debutTest = fold * taillePli;
        int finTest = debutTest + taillePli;
        double correct = 0;
        // Parcours des données de test dans ce pli
        for (int classe = 0; classe < NbClasses; classe++) {
            for (int index = debutTest; index < finTest; index++) {
                Double[] exemple = data[classe][index];
                int classeReelle = classe;
                // Calcul des distances pour l'exemple
                Double[] distances = new Double[NbClasses * (NbExLearning - taillePli)];
                ComputeDistances2(exemple, distances, debutTest, finTest);

                // Prédiction de la classe de l'exemple en utilisant k-PPV
                int classePredite = PredictClassKNearestNeighbours(distances, k);

                // Vérification si la prédiction est correcte
                if (classePredite == classeReelle) {
                    correct++;
                }
            }
        }
        // Calcul de l'accuracy pour ce pli
        double accuracyPli = correct / (taillePli * NbClasses);
        accuracyTotale += accuracyPli;
    }
    // Calcul de l'accuracy moyenne sur tous les plis
    double accuracyMoyenne = accuracyTotale / nFolds;
    return accuracyMoyenne;
}
```

Figure 5: La méthode crossValidation

Cette fonction **crossValidation**, vise à effectuer une validation croisée pour évaluer les performances de l'algorithme K-PPV. Elle prend en paramètre le nombre de voisins (k) à considérer et le nombre de plis (nFolds) pour la validation croisée. La fonction initialise la variable accuracy totale à zéro et calcule la taille de chaque pli (taillePli) en divisant le nombre total d'exemples d'apprentissage par le nombre de plis. Ensuite, pour chaque pli, elle détermine les indices de début et de fin du pli de test, puis effectue la prédiction de classe pour chaque exemple dans ce pli de test en utilisant l'algorithme K-PPV. Elle vérifie si la prédiction est correcte et incrémente le compteur correct en conséquence. Après avoir parcouru tous les plis, elle calcule la précision pour chaque pli et l'ajoute à la précision



globale. Enfin, elle divise la précision totale par le nombre de plis pour obtenir la précision moyenne de la validation croisée, qu'elle retourne.

En résumé, cette fonction réalise une validation croisée pour évaluer la précision de l'algorithme K-PPV avec un nombre spécifié de voisins.

- Résultat pour K =21 : « java kppv.KPPV 21 ».

```
Recognition rate: 0.9466666666666667
***** La Cross Validation *****
- L'accuracy donnée par la cross validation (fold =5): 0.6666666666666666
- L'accuracy donnée par la cross validation (fold =10): 0.8833333333333332
- L'accuracy donnée par la cross validation (fold =20): 0.9333333333333333
- L'accuracy donnée par la cross validation (fold =25): 0.9200000000000002
```

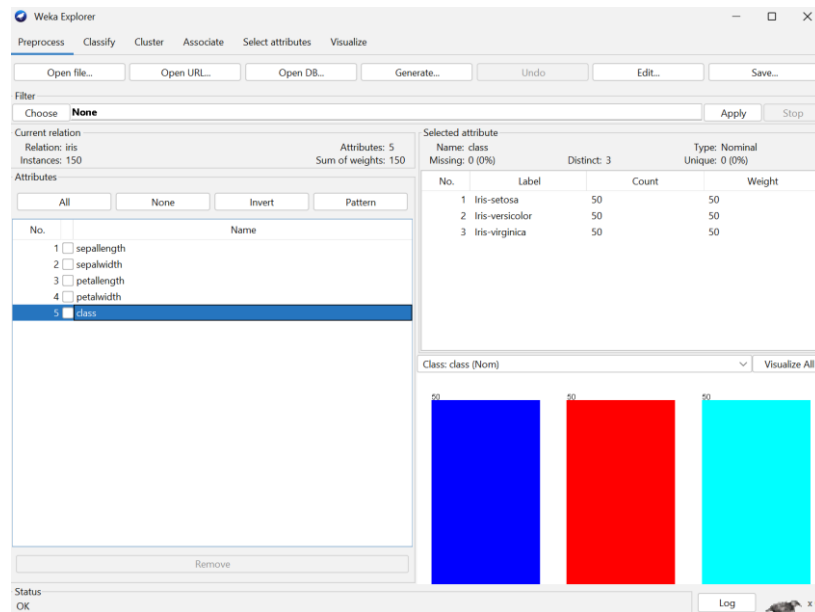
**Figure 6: la résultat données par la cross validation**

Les résultats obtenus révèlent une amélioration de la précision du modèle à mesure que le nombre de plis augmente. Nous observons une précision de 0,667 pour 5 plis, 0,883 pour 10 plis et 0,933 pour 20 plis. Cependant, lorsque le nombre de plis est porté à 25 (NFold = 25), la précision de notre modèle diminue légèrement à 0,920. Cette diminution pourrait suggérer que notre modèle commence à être surajusté aux données d'entraînement, ce qui peut affecter sa capacité à généraliser efficacement aux données inconnues.

## Deuxième étape : WEKA

### La découverte de Weka

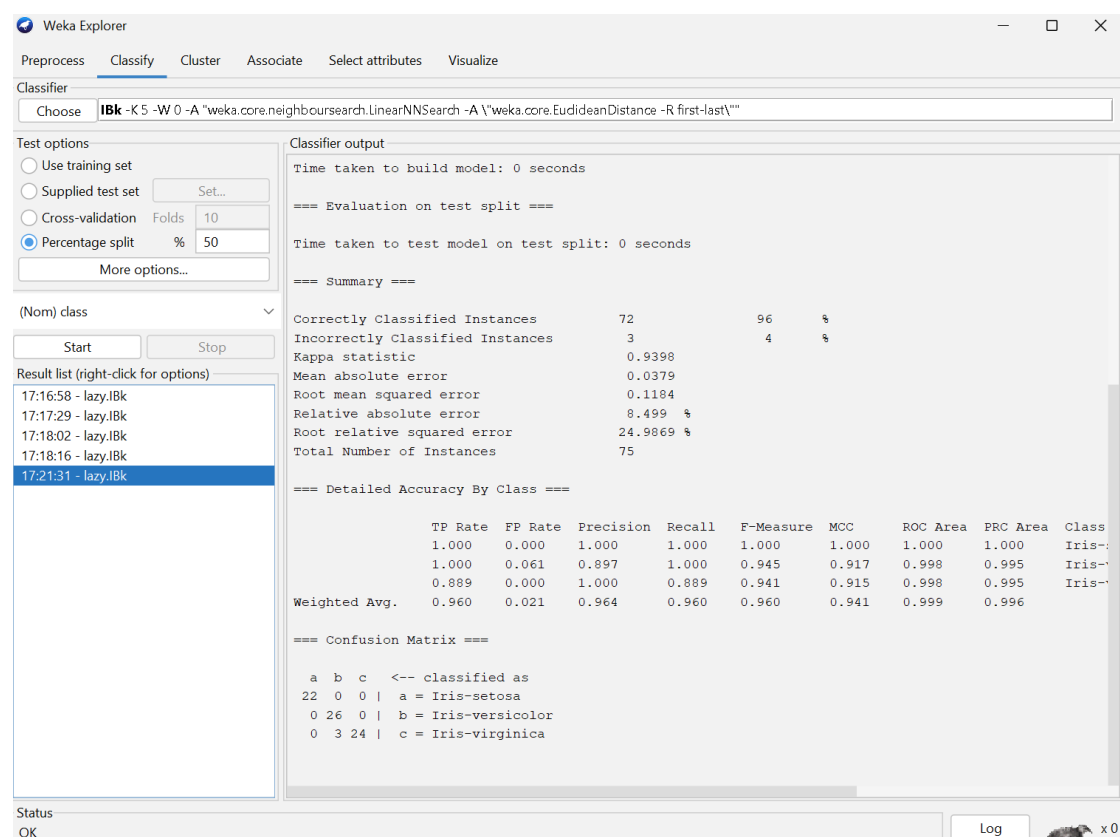
On a choisi notre fichier (.arff) pour l'analyse. En bas à droite de l'interface, on trouve les histogrammes des données, basés sur les valeurs des caractéristiques sélectionnées (par défaut, la longueur des sépales), affichés en couleur en fonction des classes d'exemples (Iris-setosa, Iris-versicolor, Iris-virginica). Vous pouvez modifier la caractéristique sélectionnée pour observer les distributions. De plus, il est possible de visualiser les données dans des espaces à deux dimensions en sélectionnant l'option « Visualiser » dans le menu.



**Figure 7:découverte de Weka**

## Classification “IBK”

On a utilisé la classification IBK avec une division de classification de 50% et on a modifié k à 5. On a également utilisé la validation croisée avec 10 fold. Dans les deux cas, on a obtenu comme résultat une matrice de confusion avec un rapport de classification comprenant des colonnes telles que TP Rate, FP Rate, Précision, Rappel, F-Mesure, Classe... On a également obtenu un résumé des résultats comprenant des données telles que les instances correctement classées, les instances incorrectement classées, l'erreur moyenne absolue, l'erreur quadratique moyenne. Voici le résultat en utilisant une division d'entraînement de 50%, illustré dans l'image suivante :



**Figure 8: Résultat avec la classification IBK(50% split)**

Analyse :

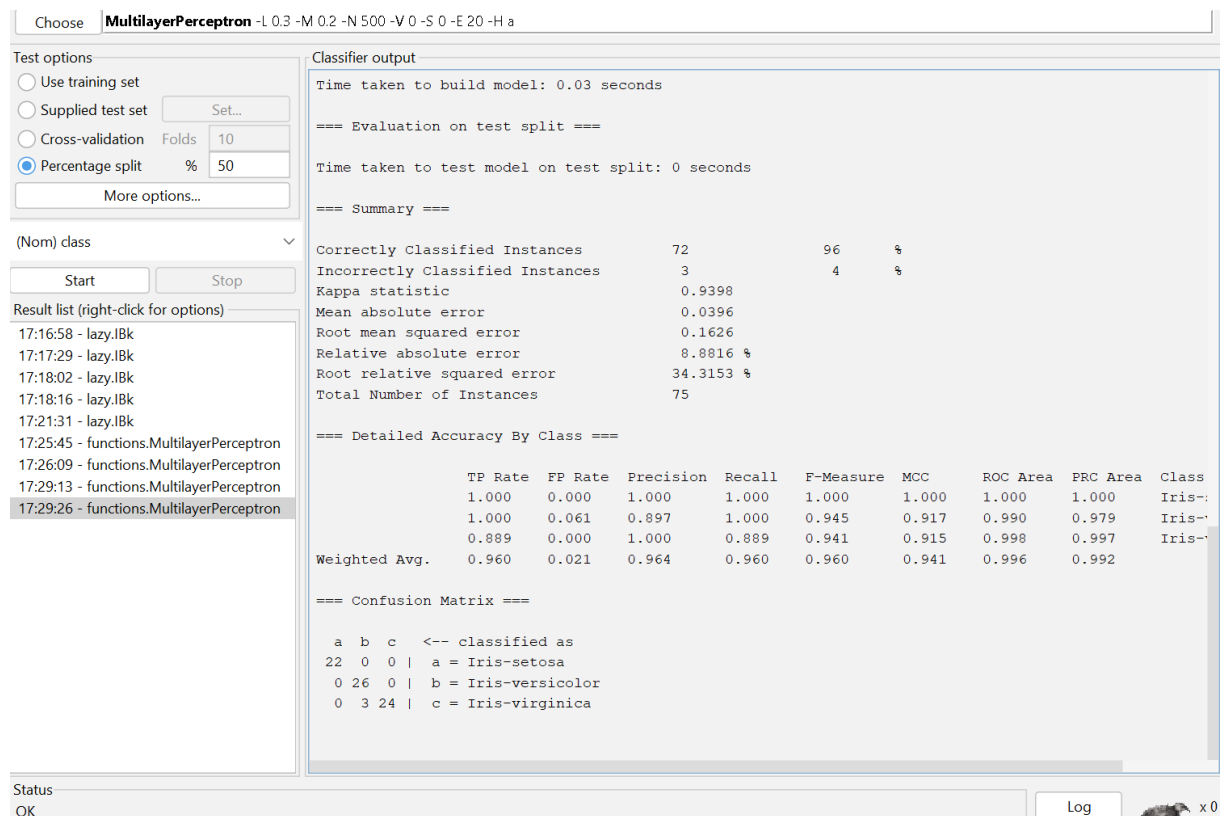
On a utilisé le classifieur IBk avec une classification basée sur les instances et une recherche des 5 plus proches voisins pour la classification. Sur l'ensemble de test, le modèle a correctement classé 96% des instances, avec seulement 4% incorrectement classées. Le coefficient kappa de 0.9398 indique une bonne fiabilité du modèle. Les mesures d'erreur moyenne absolue et d'erreur quadratique moyenne sont faibles, avec respectivement 0.0379 et 0.1184, démontrant la précision du modèle dans ses prédictions. L'analyse de l'exactitude par classe montre des performances élevées pour toutes les classes, avec des valeurs de précision, de rappel et de F-mesure proches de 1.0. La matrice de confusion révèle qu'aucune confusion n'a eu lieu entre les classes Iris-setosa et Iris-versicolor, et seulement trois confusions entre Iris-versicolor et Iris-virginica.

## Neural networks

Nous avons utilisé la classification Multilayer Perceptron avec les données fournies en TP(500 epoch...), en les divisant en deux ensembles avec une répartition de **50%**.

Dans cette analyse, le modèle Multilayer Perceptron a correctement classé 96% des instances, avec seulement 4% incorrectement classées. Le coefficient kappa de 0.9398 indique une bonne fiabilité du modèle. Les mesures d'erreur moyenne absolue et d'erreur quadratique moyenne sont respectivement de 0.0396 et 0.1626, démontrant la précision du modèle dans ses

prédictions. L'analyse de l'exactitude par classe montre des performances élevées pour toutes les classes, avec des valeurs de précision, de rappel et de F-mesure proches de 1.0.



**Figure 9: Résultat avec la classification Multilayer Perceptron (50% split)**

La matrice de confusion révèle qu'aucune confusion n'a eu lieu entre les classes Iris-setosa et Iris-versicolor, et Trois confusion entre Iris-versicolor et Iris-virginica, ce qui souligne la capacité discriminante du modèle, notées par des carrés verts dans le graphique suivant :

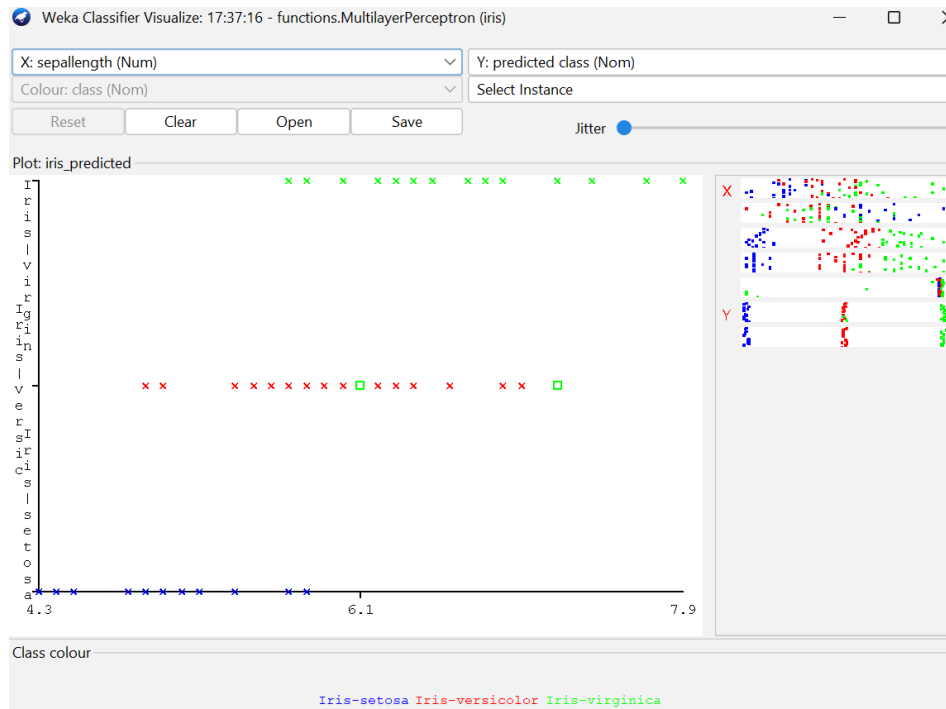
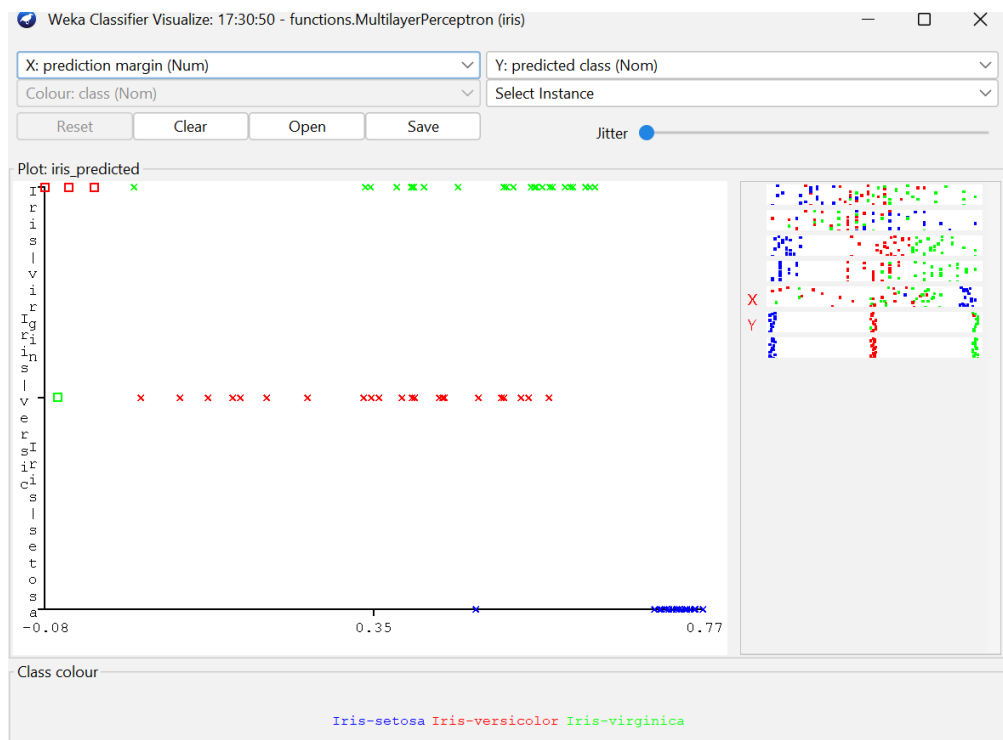


Figure 10:Le graphique de prédiction(500epochs)

## Réduction le nombre d'epochs à 25

La comparaison entre les deux résultats montre une légère diminution de la performance du modèle lorsqu'on réduit le nombre d'epochs à 25 par rapport au résultat précédent qui utilisait 500 epochs. Le modèle avec 25 epochs a correctement classé 94.67% des instances, tandis que celui avec 500 epochs en avait correctement classé 96%. De plus, l'erreur moyenne absolue et l'erreur quadratique moyenne ont augmenté dans le cas de 25 epochs, indiquant une moindre précision des prédictions.

Cependant, malgré cette diminution, le modèle avec 25 epochs reste très performant avec des valeurs élevées pour la précision, le rappel et la F-mesure pour chaque classe. La matrice de confusion montre également une capacité à discriminer efficacement entre les classes, bien que quelques confusions supplémentaires aient été observées avec le modèle à 25 epochs.



**Figure 11:Le graphique de prédiction(25 epochs)**

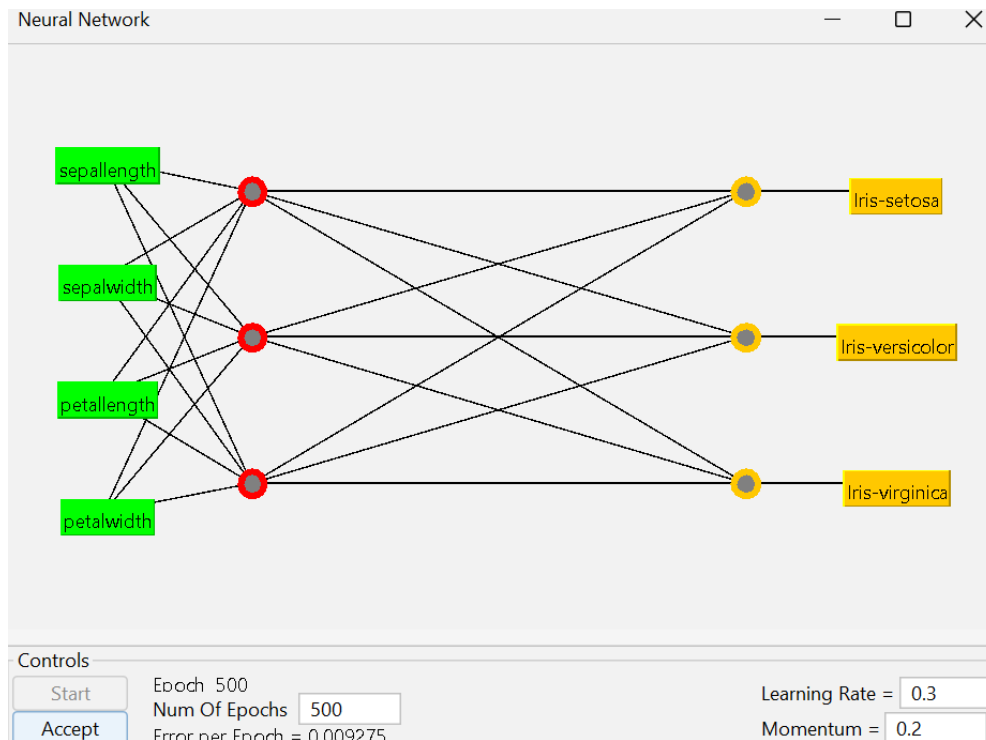
Le graphique de prédiction révèle qu'aucune confusion n'a été observée entre les classes Iris-setosa et Iris-versicolor. Cependant, quatre confusions ont été identifiées entre Iris-versicolor et Iris-virginica, notées par des carrés rouges et verts dans le graphique.

## GUI option

**Question: From this information, can you try to deduce how the concepts are modeled (which attributes are used)? Do you think we really need 3 hidden units for this iris dataset?**

Dans ce modèle, chaque attribut du jeu de données iris (sepal length, sepal width, petal length, petal width) se voit attribuer des poids pour aider à prendre des décisions. Par exemple, le nœud 3 utilise ces attributs pour déterminer la classe Iris-setosa. Les poids positifs ou négatifs associés à chaque attribut indiquent leur importance dans la prise de décision finale.

En ce qui concerne le besoin de trois unités cachées, on peut dire que cela dépend de la complexité des relations entre les caractéristiques des fleurs et leurs catégories. Si ces relations sont simples, moins d'unités cachées pourraient être nécessaires. Dans cette situation, il semble que deux unités cachées pourraient être adéquates, car le modèle parvient à de bons résultats avec seulement deux erreurs de classification.



**Figure 12:Le réseau**

### Remarque :

Dans cette situation, puisque les performances sont déjà très bonnes et que la classification est précise, l'idée d'utiliser trois unités cachées peut sembler un peu trop. Il se pourrait qu'un nombre moindre d'unités suffise pour bien saisir les schémas des données, ce qui rendrait le modèle plus simple et plus facile à comprendre.

## Tests on MNIST dataset

### L'algorithme IBK

L'algorithme IBK a été utilisé avec un paramètre K égal à 1 pour classer des instances dans un ensemble de données d'une relation entre les images d'apprentissage et leurs étiquettes. Le modèle a été construit rapidement en 0,01 seconde. Cependant, lors de l'évaluation sur un ensemble de test, il a correctement classé seulement 21,57% des instances, avec 78,43% d'instances incorrectement classées. Le coefficient kappa, mesurant l'accord entre les classifications prédites et les classifications réelles, est assez faible à 0,1248. L'erreur absolue moyenne est de 0,1569 et l'erreur quadratique moyenne est de 0,396. Ces mesures indiquent que l'algorithme n'a pas réussi à bien généraliser sur les données de test.

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier Choose **MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a**

Test options

☐ Use training set

☐ Supplied test set Set...

☐ Cross-validation Folds 10

☒ Percentage split % 50

More options...

(Nom) class

Start Stop

Result list (right-click for options)

- 17:55:36 - lazy.IBk
- 17:56:07 - lazy.IBk
- 17:59:02 - lazy.IBk
- 18:15:40 - lazy.IBk
- 18:16:37 - lazy.IBk
- 18:16:55 - lazy.IBk**
- 18:41:40 - lazy.IBk
- 18:41:52 - lazy.IBk
- 18:42:13 - functions.MultilayerPerceptron

Classifier output

Correctly Classified Instances 2157 21.57 %

Incorrectly Classified Instances 7843 78.43 %

Kappa statistic 0.1248

Mean absolute error 0.1569

Root mean squared error 0.396

Relative absolute error 87.1759 %

Root relative squared error 132.0244 %

Total Number of Instances 10000

=== Detailed Accuracy By Class ===


	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0	0.345	0.136	0.216	0.345	0.266	0.171	0.605	0.140	0
1	0.803	0.293	0.260	0.803	0.393	0.339	0.755	0.231	1
2	0.079	0.031	0.227	0.079	0.118	0.079	0.524	0.113	2
3	0.114	0.074	0.147	0.114	0.128	0.045	0.520	0.106	3
4	0.242	0.111	0.192	0.242	0.214	0.119	0.566	0.121	4
5	0.076	0.058	0.115	0.076	0.092	0.023	0.509	0.091	5
6	0.112	0.054	0.180	0.112	0.138	0.072	0.529	0.105	6
7	0.238	0.097	0.219	0.238	0.228	0.136	0.571	0.131	7
8	0.000	0.000	?	0.000	?	?	0.500	0.097	8
9	0.053	0.022	0.215	0.053	0.084	0.060	0.516	0.107	9
Weighted Avg.	0.216	0.091	?	0.216	?	?	0.563	0.126	

=== Confusion Matrix ===

	a	b	c	d	e	f	g	h	i	j	<-- classified as
338	134	38	76	113	116	53	85	0	27	1	a = 0
38	911	17	20	45	34	11	38	0	21	1	b = 1
162	310	82	80	122	84	103	74	0	15	1	c = 2
138	383	20	115	93	92	29	109	0	31	1	d = 3
118	272	28	64	238	26	68	145	0	23	1	e = 4
135	277	18	70	113	68	68	118	0	25	1	f = 5
169	262	53	98	98	62	107	98	0	11	1	g = 6
140	260	53	82	136	24	26	245	0	22	1	h = 7

Status

Building model on training data...

Log  x1

la précision par classe révèle des performances variables. Par exemple, la classe 1 a une précision de 80,3%, tandis que la classe 8 n'a pas pu être évaluée. Les taux de vrais positifs (TP Rate) varient considérablement d'une classe à l'autre, allant de 0% pour la classe 8 à 80,3% pour la classe 1. Les mesures de précision, de rappel et de F-mesure montrent également des écarts significatifs entre les classes, indiquant des difficultés spécifiques dans la classification de certaines classes par rapport à d'autres.