UNIVERSITÉ
LAVAL

TP3 Techniques avancées en Intelligence Artificielle

Fait par:

MAAOU Marouane IFT-7025

1 Arbres de Décision

1.1 Entraînement

Dans les figures ci-dessous, on représente la courbe d'apprentissage pour faire le *sanity-check* de notre algorithme d'apprentissage sur chaque jeu de donnée.

1.1.1 IRIS

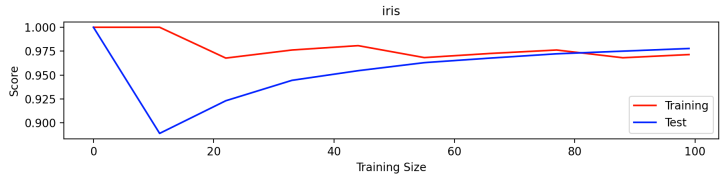

Figure 1: Courbe d'apprentissage en augmentant la taille du jeu de donnée IRIS avec 10

1.1.2 WINE

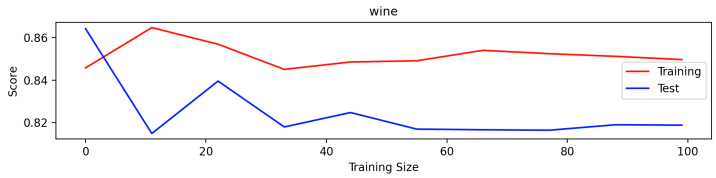

Figure 2: Courbe d'apprentissage en augmentant la taille du jeu de donnée WINE avec 10

1.1.3 ABALONES

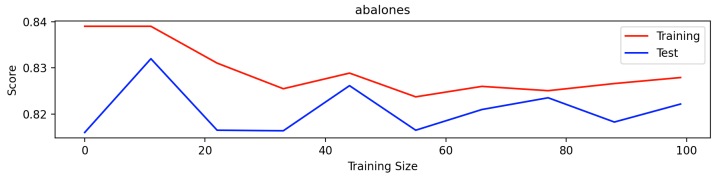

Figure 3: Courbe d'apprentissage en augmentant la taille du jeu de donnée ABALONES avec 10

On remarque que généralement au début c.à.d avec une taille petite, l'accuracy sur l'ensemble d'entrainement est grande alors qu'elle est moins bonne sur l'ensemble de test. Cela dû au fait que l'algorithme n'arrive pas encore à généraliser sur l'ensemble de test puisqu'il n'y a pas suffisamment de données d'entrainement, mais on remarque que plus la taille des données augmente l'accuracy du jeu de test s'améliore.

1.2 Évaluation

On évalue notre modèle sur l'ensemble de test de chaque dataset.

1.2.1 IRIS

Figure 4: Les métriques de notre modèle sur les données de test IRIS

1.2.2 WINE

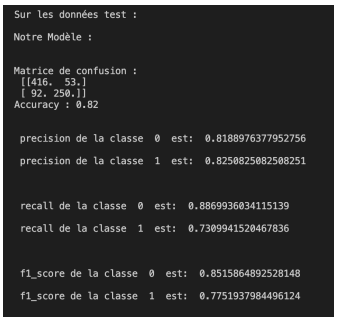

Figure 5: Les métriques de notre modèle sur les données de test WINE

1.2.3 ABALONES

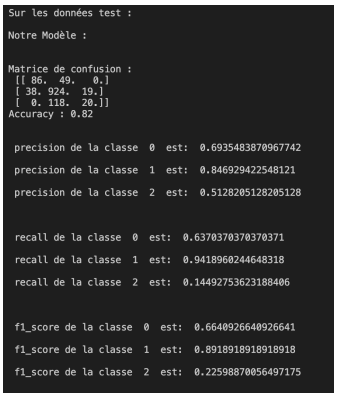

Figure 6: Les métriques de notre modèle sur les données de test ABALONES

On remarque que notre modèle en général performe bien sur les datasets IRIS et WINE, mais il trouve des difficultés avec ABALONES surtout les faux négatives

de la classe 1 (nombre d'anneaux entre 8 et 14).

On visualise l'arbre de décision obtenu par notre modèle avec le jeu de données IRIS.

```
 Decision Tree de iris  est
Colone: 2 seuil <= 1.9 gain d'information 0.94
--- Gauche:0.0
--- Droit:Colone: 2 seuil <= 4.7 gain d'information 0.73
------ Gauche:1.0
------ Droit:Colone: 2 seuil <= 5.1 gain d'information 0.14
---------- Gauche:Colone: 3 seuil <= 1.7 gain d'information 0.21
-------------------- Gauche:1.0
-------------------- Droit:2.0
---------- Droit:2.0
```

Figure 7: Arbre de décision du jeu de données IRIS.

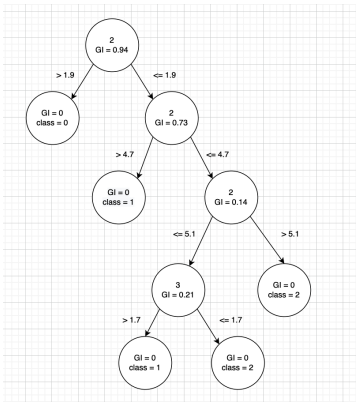

Figure 8: Arbre de décision du jeu de données IRIS déssinée.

NB: {'*Iris-setosa*': 0, '*Iris-versicolor*' : 1, '*Iris-virginica*' : 2}

On évalue avec la bibliothèque *sklearn* l'ensemble de test de chaque dataset.

1.2.4 IRIS

```
 Scikit-learn:

Matrice de confusion :
 [[15  0  0]
 [ 0 15  4]
 [ 0  0 11]]
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000        15
           1     1.0000    0.7895    0.8824        19
           2     0.7333    1.0000    0.8462        11

    accuracy                         0.9111        45
   macro avg     0.9111    0.9298    0.9095        45
weighted avg     0.9348    0.9111    0.9127        45
```

Figure 9: Les métriques de sklearn sur les données de test IRIS

1.2.5 WINE

```
Scikit-learn:

Matrice de confusion :
 [[405  64]
 [ 64 278]]
              precision    recall  f1-score   support

           0     0.8635    0.8635    0.8635       469
           1     0.8129    0.8129    0.8129       342

    accuracy                         0.8422       811
   macro avg     0.8382    0.8382    0.8382       811
weighted avg     0.8422    0.8422    0.8422       811
```

Figure 10: Les métriques de sklearn sur les données de test WINE

1.2.6 ABALONES

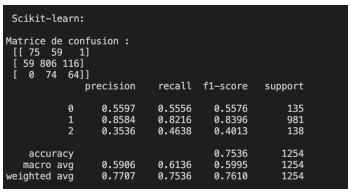

Figure 11: Les métriques de sklearn sur les données de test ABALONES

On remarque aussi que notre modèle de sklearn en général performe bien sur les datasets IRIS et WINE, mais il résulte des faux négatives de la classe 2 (nombre d'anneaux > 14) pour ABALONES.

2 Réseaux de Neurones Artificiels

2.1 Initialisation des poids du réseau de neurones

NB: Dans toutes les partie, on normalise nos datasets min-max parceque les réseaux de neurones nécéssitent des données normalisées.

L'initialisation de tous les poids du RN à la même valeur (zero ou autre) n'est pas une bonne pratique. Pour cela, nous avons choisit de les initialiser avec une distribution *uniform* entre -1 et 1 car il est important d'utiliser de petites valeurs pour les poids de sorte que, au début de l'apprentissage, le réseau fonctionne en mode linéaire, puis qu'il augmente les valeurs de ses poids afin d'ajuster les données avec suffisamment de précision. Les deux figures ci-dessous nous illustre les deux types d'initialisation.

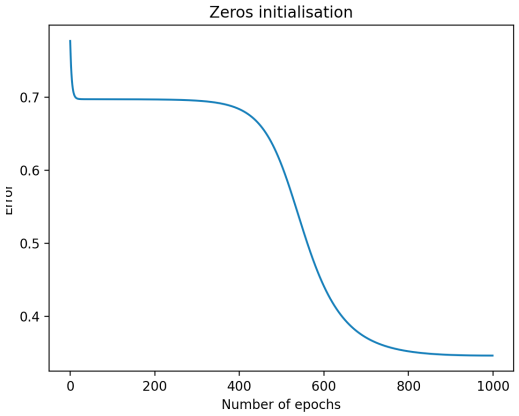

Figure 1: Initialisation de tous les poids à zero

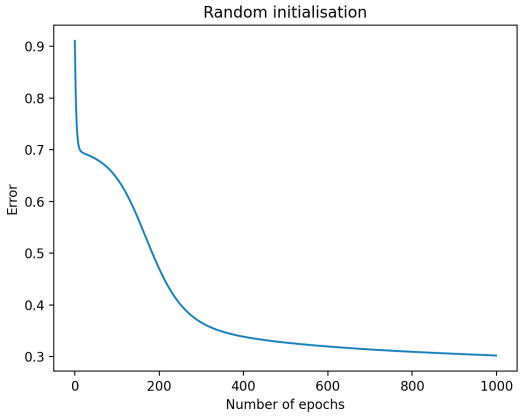

Figure 2: Initialisation des les poids avec des valeurs aléatoires entre -1 et 1

On remarque que l'initialisation de tous les poids à zero converge à un minimum d'une façon plus lente (à l'epoch 800) que l'initialisation aléatoire (à l'epoch 400). En effet, dans le cas de l'initialisation zero le modèle prend beaucoup de temps pour ajuster les poids ce qui explique la stabilite de la courbe entre l'epoch 0 et l'epoch 500 dans la fig. 1 ainsi cela peut être une problématique lors du manque de ressources de computation.

On entraine les datasets *Iris*, *wine*, *Abalones* avec une incrémenation de 10 des tailles des données avec une initialisation à zero et aléatoire en fonction de l'*accuracy* avec un *learning_rate* $= 0.01$, 5 unitées dans la couche cachée et 50 *epochs*.

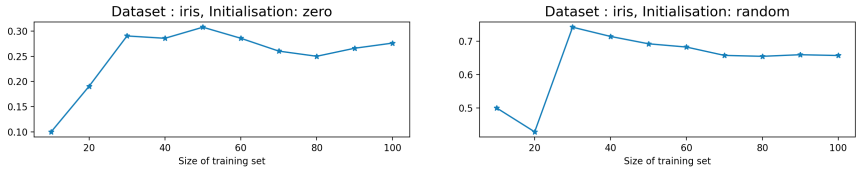

Figure 3: Initialisation des les poids avec le jeu donnée Iris

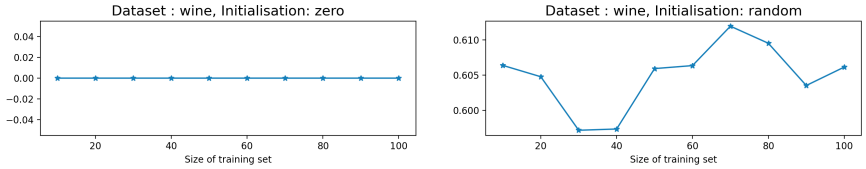

Figure 4: Initialisation des les poids avec le jeu donnée Wine

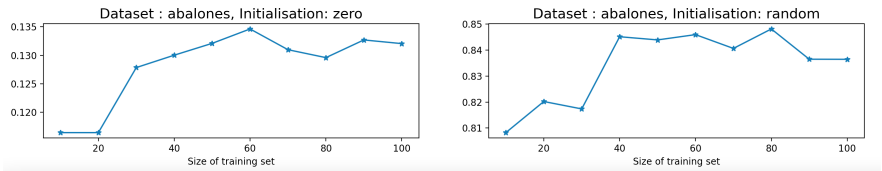

Figure 5: Initialisation des les poids avec le jeu donnée Abalones

Selon ces figures on remarque que la performance est mieux avec une initialisation aléatoire.

2.2 Entraînement et évaluation

On entraine nos datasets avec un *learning_rate* = 0.01, 15 unitées dans la couche cachée et 1000 *epochs*.

2.2.1 IRIS

Figure 6: Les métriques de notre modèle sur les données d'entrainement IRIS

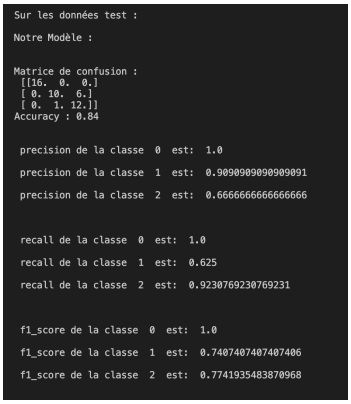

Figure 7: Les métriques de notre modèle sur les données de test IRIS

2.2.2 WINE

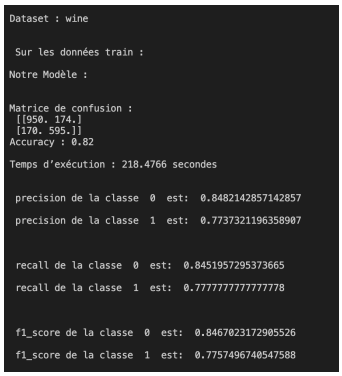

Figure 8: Les métriques de notre modèle sur les données d'entrainement WINE

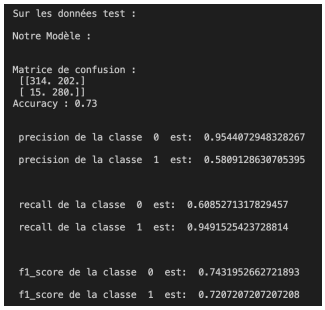

Figure 9: Les métriques de notre modèle sur les données de test WINE

2.2.3 ABALONES

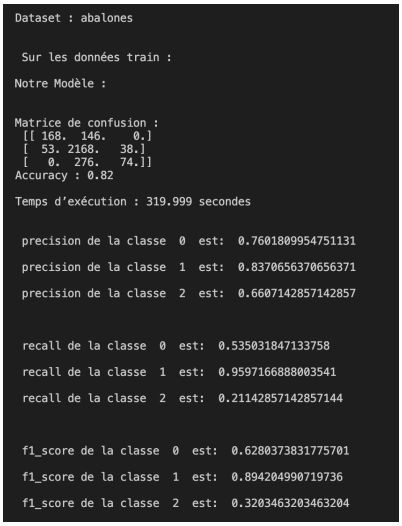

Figure 10: Les métriques de notre modèle sur les données d'entrainement ABALONES

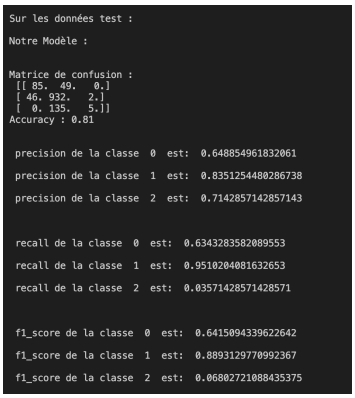

Figure 11: Les métriques de notre modèle sur les données de test ABALONES

Nous remarquons que notre modèle pérforme bien en général sur nos datasets, mais nous

2.3 Recherche d'hyperparamètres

Dans cette partie, on utilise **MLPClassifier** de la bibliothèque *sklearn*.

2.3.1 Nombre de unitées

On entraine notre réseau en utilisant la validation croisée avec un nombre de folds = 10, un *learning_rate* par defaut de 0.001, *max_iteration* de 300 sur un interval de 5 à 45 unitées avec un pas de 5 sur nos datasets.

+ IRIS

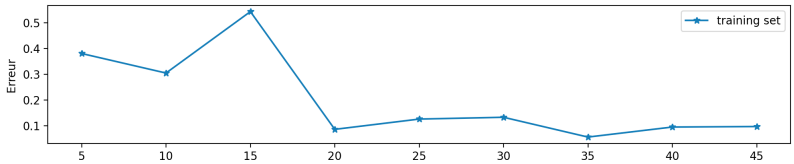

Figure 12: Erreur moyenne sur l'ensemble d'entrainement du IRIS selon le nombre des unitées.

On remarque que 35 est le meilleur nombre des unitées dans la couche cachée pour IRIS.

+ WINE

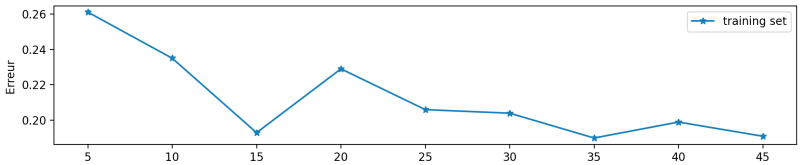

Figure 13: Erreur moyenne sur l'ensemble d'entrainement du WINE selon le nombre des unitées.

On remarque qu'aussi 35 est le meilleur nombre des unitées dans la couche cachée pour WINE.

+ ABALONES

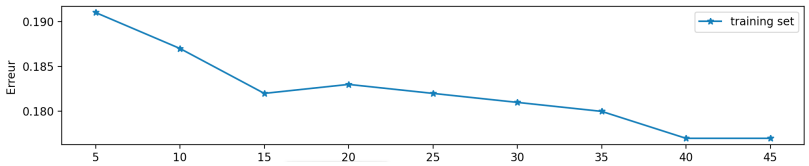

Figure 14: Erreur moyenne sur l'ensemble d'entrainement du ABALONES selon le nombre des unitées.

On remarque que 40 est le meilleur nombre des unitées dans la couche cachée pour ABALONES.

2.3.2 Nombre de couches cachées

On sauvegarde les nombres de neuronnes optimal pour chaque dataset qu'on a trouvé dans la partie précédente et on cherche dans cette partie le nombre de couches cachées optimal pour chaque dataset. On entraine avec les mêmes paramètres que la partie précédante sur un interval de nombre de couches cachées entre 1 et 9 avec un pas de 2.

+ IRIS

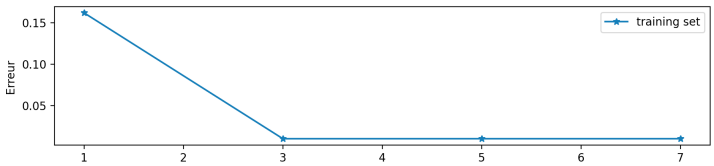

Figure 15: Erreur moyenne sur l'ensemble d'entrainement du IRIS selon le nombre des couches cachées.

On remarque que l'erreur moyenne devient optimal pour plus que 3 couches cachées pour IRIS.

+ WINE

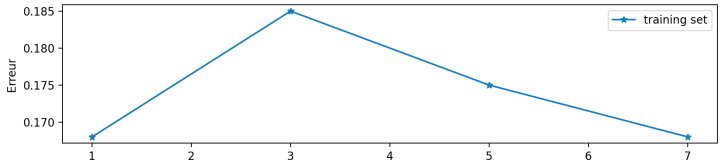

Figure 16: Erreur moyenne sur l'ensemble d'entrainement du WINE selon le nombre des couches cachées.

On remarque que l'erreur moyenne est optimale avec une seul couche cachée pour WINE, c'est un résultat logique puisque on a juste 2 classes.

+ ABALONES

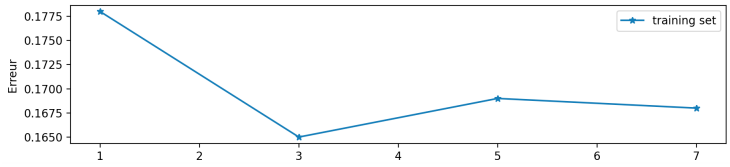

Figure 17: Erreur moyenne sur l'ensemble d'entrainement du ABALONES selon le nombre des couches cachées.

On remarque que l'erreur moyenne est optimale pour 3 couches cachées pour ABALONES.

Vanishing gradient

Au fur et à mesure que des couches cachées (utilisant certaines fonctions d'activation) sont ajoutées aux réseaux de neurones, les gradients de la fonction de perte approchent de zéro, ce qui rend le réseau difficile à s'entrainer (source *TowardScience*). Certaines fonctions d'activation, comme la fonction *sigmoïde*, rend un grand espace d'entrée dans un petit espace d'entrée entre 0 et 1. Par conséquent, ce grand changement dans l'entrée de la fonction sigmoïde entraînera un petit changement dans la sortie. Par conséquent, la dérivée devient proche de 0.

Dans les figures 15, 16 et 17 on ne voit pas ce phénomène puisque on entraîne sur un petit interval de nombre des couches cachées. Pour observer ce phénomène, on entraine le jeu de donnée IRIS sur un plus large interval.

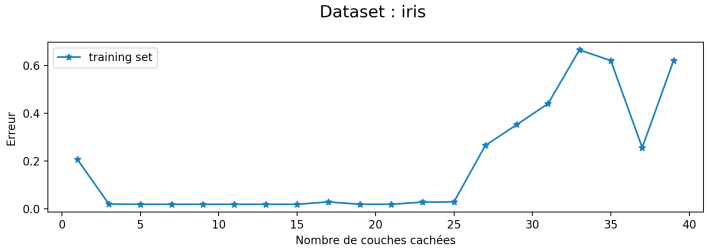

Figure 18: Erreur moyenne sur l'ensemble d'entrainement du WINE selon le nombre des couches cachées.

On observe qu'après 25 couches cachées, l'erreur moyenne augmente soudainement, on peut expliquer cela par le *vanishing gradient*.

2.4 Entraînement et Test

En utilisant les différents hyperparamètres touver dans les sections précédentes pour chaque dataset, on entraine à nouveau en utilisant la validation croisée à

10 flods et en conservant le même *learning_rate* mais cette fois au lieu de 1000 *epochs*, on utilise 100 *epochs*.

2.4.1 IRIS

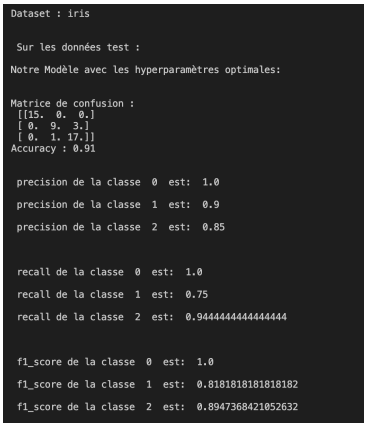

Figure 19: Les métriques de notre modèles optimisé sur les données de test IRIS

2.4.2 WINE

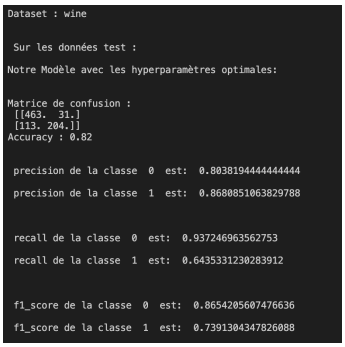

Figure 20: Les métriques de notre modèle optimisé sur les données de test WINE

2.4.3 ABALONES

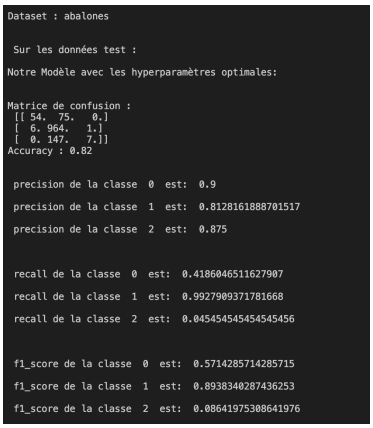

Figure 21: Les métriques de notre modèle optimisé sur les données de test ABALONES

On remarque les métriques de chaque dataset sont augmentées significativement en utilisant moins de 10x le nombre des epochs qu'auparavant. Ce qui montre l'importance de la recherche des hyperparamètres optimals pour améliorer le temps d'entrainement.

3 Tableau récapitulatif

3.1 IRIS

	Knn					Naive Bayes				
	precision	recall	F1-score	accuracy	Time (s)	precision	recall	F1-score	accuracy	Time (s)
Classe 0	100%	100%	100%	93%	2.90 x 10-5	100%	100%	100%	87%	4.21 x 10-4
Classe 1	89%	94%	91%			78%	88%	82%		
Classe 2	94%	88%	91%			86%	75%	80%		

	Arbre de décision					Réseaux de neurones				
	précision	recall	F1-score	accuracy	Time (s)	précision	recall	F1-score	accuracy	Time (s)
Class 0	100%	100%	100%	98%	9.32 x 10-2	100%	100%	100%	84%	11.35
Class 1	100%	95%	98%			91%	63%	75%		
Class 2	92%	100%	96%			67%	93%	78%		

3.2 WINE

	Knn					Naive Bayes				
	precision	recall	F1-score	accuracy	Time (s)	precision	recall	F1-score	accuracy	Time (s)
Classe 0	78%	80%	80%	75%	2.86 x 10-6	87%	76%	81%	78%	1.14 x 10-3
Classe 1	70%	68%	69%			69%	82%	75%		

	Arbre de décision					Réseaux de neurones				
	précision	recall	F1-score	accuracy	Time (s)	précision	recall	F1-score	accuracy	Time (s)
Class 0	82%	89%	86%	82%	12.38	96%	61%	75%	73%	218.48
Class 1	83%	74%	78%			59%	95%	73%		

3.3 ABALONES

	Knn					Naive Bayes				
	precision	recall	F1-score	accuracy	Time (s)	precision	recall	F1-score	accuracy	Time (s)
Classe 0	69%	52%	59%	81%	1.66 x 10-6	43%	92%	58%	57%	1.23 x 10-3
Classe 1	84%	93%	88%			86%	53%	66%		
Classe 2	59%	29%	39%			22%	52%	31%		

	Arbre de décision					Réseaux de neurones				
	précision	recall	F1-score	accuracy	Time (s)	précision	recall	F1-score	accuracy	Time (s)
Class 0	70%	64%	67%	82%	43.55	65%	64%	64%	81%	320
Class 1	85%	95%	90%			84%	96%	89%		
Class 2	52%	15%	23%			72%	4%	7%		

4 Conclusion

Dans le Tp3 et ce travail, nous avons implémentée quatre classifeurs, le naif bayes, le k plus proches voisins, les arbres de décisions et les réseaux de neurones que nous les avons entrainées sur trois ensembles de données differents. Nous avons comparée les résultats obtenus avec ceux obtenus par chaque classifieurs.

Ces travaux nous a permi de consolider plus les notions vu theoriquement en classes, sutout la partie des arbres des decisions qui est théoriquement facile mais un peu plus difficile en code (je suis un peu prudent des programmes qui contient des choses récursives :')).