UNIVERSITÉ
LAVAL

TP3 Techniques avancées en Intelligence Artificielle

Fait par:

MAAOU Marouane IFT-7025

1 Question 1: les métriques de classification

1.1 Matrice de confusion

En apprentissage automatique supervisé, la matrice de confusion est une matrice qui mesure la qualité d'un système de classification. L'idée générale est de compter le nombre de fois où les instances de la classe A sont classées dans la classe B. Pour calculer la matrice de confusion, on doit d'abord disposer d'un ensemble de prédictions, afin qu'elles puissent être comparées aux cibles réelles. Chaque ligne d'une matrice de confusion représente une classe réelle, tandis que chaque colonne représente une classe prédite. Les résultats d'une matrice de confusion sont classés en quatre grandes catégories :

- **Faux positif (FP)**: on classifie un patient comme bien portant alors qu'il est bien malade.
- **Faux négatif (FN)**: on classifie un patient comme bien malade alors qu'il est bien portant.
- **Vrai positif (TP)**: on classifie un patient comme malade et il est effectivement malade.
- **Vrai négatif (TN)**: on classifie un patient comme bien portant et il est bien portant.

1.2 Exactitude

L'exactitude indique le pourcentage de bonnes prédictions. C'est un très bon indicateur parce qu'il est très simple à comprendre.

Elle est calculée par la formule

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

1.3 Précision

La précision donne le pourcentage de réponses correctes, ce qui permet de répond à la question suivante : sur tous les enregistrements positifs prédits, combien sont réellement positifs ?

Elle est calculée par la formule

$$Precision = \frac{TP}{TP+FP}$$

1.4 Rappel

Le rappel donne le pourcentage des réponses correctes qui sont données, ce qui permet de répondre à la question suivante : sur tous les enregistrements positifs, combien ont été correctement prédits ?

Elle est calculée par la formule

$$Rappel = \frac{TP}{TP+FN}$$

1.5 F1-score

F1-score est une moyenne harmonique de la précision et du rappel. Il équivaut au double du produit de ces deux paramètres sur leur somme. Sa valeur est maximale lorsque le rappel et la précision sont équivalents.

Elle est calculée par la formule

$$F1\text{-}score = 2.\frac{precision.rappel}{precision+rappel}$$

2 Question 2: K plus proches voisins (K-nearest neighbors)

2.1 La métrique de distance

Pour que l'algorithme fonctionne au mieux sur un ensemble de données, nous devons choisir la métrique de distance la plus appropriée. Il existe de nombreuses mesures de distance différentes comme la distance de **Minkowski**, **Manhattan**, **Cosine** (similarité). Pour ce travaille, on a utilisé la distance **Euclidienne** qui est la plus populaire et qui est potentiellement bonne pour la plupart des ensembles des données car il donne la distance à vol d'oiseau.

2.2 Pseudo-code

Pour une entrée x, on calcule la distance euclidienne entre x et tout nos données d'entrainement et on les ajoute dans une liste. On trie cette liste on ordre croissant et on choisit les K premiers éléments de la liste. On assigne à x la classe dominante des K éléments. La figure suivante présente un pseudo-code de l'algorithme K-NN.

```
distances = []
for i in range(len(self.trainSet)):
    distance = self.dist_euclidien(self.trainSet[i], x)
    distances.append((self.train_labels[i], distance))
distances.sort(key=lambda tuple: tuple[1]) # sort tuple ascending by distance
classes = []
for i in range(self.k):
    classes.append(distances[i][0])
pred = max(set(classes), key=classes.count)
return pred
```

2.3 Evaluation avec notre implémentation avec une validation croisée sur un intervalle [1, 24]

En utilisant la validation croisée pour chercher le nombre des voisins optimal sur chaque ensemble de donnée, on a trouver que k=10 est le meilleur pour l'ensemble de donnée **Iris**, k=1 pour **Wine** et k=7 pour **Abalones**.

```
le meilleur k pour IRIS est :  10
le meilleur k pour Wine est :  1
le meilleur k pour Abalone est :  7
```

2.3.1 Iris

```
I-1-2) Les métriques avec notre modèle KNN avec les données de test IRIS:
la matrice de confusion est:
  [[19.  0.  0.]
 [ 0. 16.  1.]
 [ 0.  0.  9.]]
------- l'accuracy du modèle est:  97.78 %
+++++++ Precision de la classe  0  est:  100.0 %
+++++++ Precision de la classe  1  est:  100.0 %
+++++++ Precision de la classe  2  est:  90.0 %
******* Recall de la classe  0  est:  100.0 %
******* Recall de la classe  1  est:  94.12 %
******* Recall de la classe  2  est:  100.0 %
####### f1 score de la classe  0  est:  100.0 %
####### f1 score de la classe  1  est:  96.97 %
####### f1 score de la classe  2  est:  94.74 %
temps d'execution de KNN sur Iris est:  9.5367431640625e-07  S
```

2.3.2 Wine

```
I-2-2) Les métriques avec notre modèle KNN avec les données de test WINE:
la matrice de confusion est:
  [[387.  94.]
 [107. 223.]]
------- l'accuracy du modèle est:  75.22 %
+++++++ Precision de la classe  0  est:  78.34 %
+++++++ Precision de la classe  1  est:  70.35 %
******* Recall de la classe  0  est:  80.46 %
******* Recall de la classe  1  est:  67.58 %
####### f1 score de la classe  0  est:  79.38 %
####### f1 score de la classe  1  est:  68.93 %
temps d'execution de KNN sur Wine est:  9.5367431640625e-07  S
```

2.3.3 Abalones

```
I-3-2) Les métriques avec notre modèle KNN avec les données de test ABALONES:
la matrice de confusion est:
  [[ 79.  74.   0.]
 [ 35. 899.  28.]
 [  0.  98.  41.]]
------- l'accuracy du modèle est:  81.26 %
+++++++ Precision de la classe  0  est:  69.3 %
+++++++ Precision de la classe  1  est:  83.94 %
+++++++ Precision de la classe  2  est:  59.42 %
******* Recall de la classe  0  est:  51.63 %
******* Recall de la classe  1  est:  93.45 %
******* Recall de la classe  2  est:  29.5 %
####### f1 score de la classe  0  est:  59.18 %
####### f1 score de la classe  1  est:  88.44 %
####### f1 score de la classe  2  est:  39.42 %
temps d'execution de KNN sur Abalones est:  3.0994415283203125e-06  S
```

2.4 Evaluation avec Sklearn

2.4.1 Iris

```
I) Scikit-learn KNN pour IRIS:

Matrice de confusion :
 [[19  0  0]
 [ 0 16  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.94      0.97        17
           2       0.90      1.00      0.95         9

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.97        45
weighted avg       0.98      0.98      0.98        45
```

2.4.2 Wine

```
I) Scikit-learn KNN pour Wine:

Matrice de confusion :
 [[387  94]
 [107 223]]
              precision    recall  f1-score   support

           0       0.78      0.80      0.79       481
           1       0.70      0.68      0.69       330

    accuracy                           0.75       811
   macro avg       0.74      0.74      0.74       811
weighted avg       0.75      0.75      0.75       811
```

2.4.3 Abalones

```
I) Scikit-learn KNN pour ABALONES:

Matrice de confusion :
 [[ 79  74   0]
 [ 35 899  28]
 [  0  98  41]]
              precision    recall  f1-score   support

           0       0.69      0.52      0.59       153
           1       0.84      0.93      0.88       962
           2       0.59      0.29      0.39       139

    accuracy                           0.81      1254
   macro avg       0.71      0.58      0.62      1254
weighted avg       0.79      0.81      0.79      1254
```

On remarque que notre implémentation de l'algorithme k plus proches voisins donne les mêmes résultats que celle de sklearn.

3 Question 3: Naïve Bayésienne

3.1 Pseudo-code

Tout d'abord, on calcule la moyenne, écart-type et la probabilté à priori pour chaque classe.

```
for classe in classes:
    means[classe] = np.mean(train[train_labels == classe , :] , axis = 0)
    stds[classe] = np.std(train[train_labels == classe , :] , axis = 0)
    classe_priors[classe] = train[train_labels == classe , :].shape[0] / train.shape[0]
```

Ensuite, on calcule les probablités en utilisant la densité de probabilité de la loi normale, puis on calcule les prédictions (vraisemblence) de chaque classes et on choisit la plus grande prédictions.

```
predictions = {}
for classe in np.unique(self.train_labels):
    poste = 1
    for i in range(len(x)):
        poste = poste * self.normal_distrubution(x[i], self.means[classe][i], self.stds[classe][i])
    predictions[classe] = self.classe_priors[classe] * poste
return max(predictions, key=predictions.get)
```

3.2 Evaluation avec notre implémentation

3.2.1 Iris

```
I-1-2) Les métriques avec notre modèle bayes naif avec les données de test IRIS:
la matrice de confusion est:
  [[13.  0.  0.]
 [ 0. 14.  2.]
 [ 0.  4. 12.]]
------- l'accuracy du modèle est:  86.67 %
+++++++ Precision de la classe  0  est:  100.0 %
+++++++ Precision de la classe  1  est:  77.78 %
+++++++ Precision de la classe  2  est:  85.71 %
******* Recall de la classe  0  est:  100.0 %
******* Recall de la classe  1  est:  87.5 %
******* Recall de la classe  2  est:  75.0 %
####### f1 score de la classe  0  est:  100.0 %
####### f1 score de la classe  1  est:  82.35 %
####### f1 score de la classe  2  est:  80.0 %
temps d'execution de BAYES NAIF sur Iris est:  0.00042128562927246094  S
```

3.2.2 Wine

```
I-2-2) Les métriques avec notre modèle bayes naif avec les données de test WINE:
la matrice de confusion est:
  [[369. 122.]
 [ 57. 263.]]
------- l'accuracy du modèle est:  77.93 %
+++++++ Precision de la classe  0  est:  86.62 %
+++++++ Precision de la classe  1  est:  68.31 %
******* Recall de la classe  0  est:  75.15 %
******* Recall de la classe  1  est:  82.19 %
####### f1 score de la classe  0  est:  80.48 %
####### f1 score de la classe  1  est:  74.61 %
temps d'execution de BAYES NAIF sur Wine est:  0.0011410713195800781  S
```

3.2.3 Abalones

```
I-3-2) Les métriques avec notre modèle bayes naif avec les données de test ABALONES:
la matrice de confusion est:
  [[121.  11.   0.]
 [159. 516. 292.]
 [  2.  72.  81.]]
------- l'accuracy du modèle est:  57.26 %
+++++++ Precision de la classe  0  est:  42.91 %
+++++++ Precision de la classe  1  est:  86.14 %
+++++++ Precision de la classe  2  est:  21.72 %
******* Recall de la classe  0  est:  91.67 %
******* Recall de la classe  1  est:  53.36 %
******* Recall de la classe  2  est:  52.26 %
####### f1 score de la classe  0  est:  58.45 %
####### f1 score de la classe  1  est:  65.9 %
####### f1 score de la classe  2  est:  30.68 %
temps d'execution de BAYES NAIF sur Abalones est:  0.0012369155883789062  S
```

3.3 Evaluation avec Sklearn

3.3.1 Iris

```
I) Scikit-learn BAYES NAIF pour IRIS:

Matrice de confusion :
 [[13  0  0]
 [ 0 14  2]
 [ 0  4 12]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.78      0.88      0.82        16
           2       0.86      0.75      0.80        16

    accuracy                           0.87        45
   macro avg       0.88      0.88      0.87        45
weighted avg       0.87      0.87      0.87        45
```

3.3.2 Wine

```
I) Scikit-learn BAYES NAIF pour Wine:

Matrice de confusion :
 [[373 118]
 [ 57 263]]
              precision    recall  f1-score   support

           0       0.87      0.76      0.81       491
           1       0.69      0.82      0.75       320

    accuracy                           0.78       811
   macro avg       0.78      0.79      0.78       811
weighted avg       0.80      0.78      0.79       811
```

3.3.3 Abalones

```
I) Scikit-learn BAYES NAIF pour ABALONES:

Matrice de confusion :
 [[121  11   0]
 [159 516 292]
 [  2  72  81]]
              precision    recall  f1-score   support

           0       0.43      0.92      0.58       132
           1       0.86      0.53      0.66       967
           2       0.22      0.52      0.31       155

    accuracy                           0.57      1254
   macro avg       0.50      0.66      0.52      1254
weighted avg       0.74      0.57      0.61      1254
```

On remarque aussi que notre implémentation de l'algorithme de Naive Bayes donne les mêmes résultats que celle de sklearn.

4 Tableau récapitulatif

4.1 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%		

4.2 Wine

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

4.3 Abalones

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

Selon ces tableaux, on peut conclure que l'algorithme knn est mieux que naive bayes en temps d'entrainement ainsi il est mieux en accuracy pour les données Iris et Abalones. Pour le jeu de donnée Wine, naif bayes est plus performant que knn.

5 Conclusion

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

La difcultée rencontrée est d'implémenter la fonction qui nous permet de faire la validation croisée en tenant compte son temps de calcul (complexité). Nous avons pu implémenter cette fonction avec une complexité temporelle de $O(n.m)$ avec n est le nombre des folds et m la taille de notre ensemble de donnée.