IFT-4102 et IFT-7025

Techniques avancées en Intelligence Artificielle

Rapport du TP3

Équipe 20

Membres de l'équipe :

Emma KILBERTUS - 537305286

Maxence QUELENNEC - 536899537

Estrella PAOLI - 537026544

Travail présenté à :

Brahim Chaib-draa

Remise :

01 décembre 2024

I. Définitions des métriques

- *Précision* : La précision mesure le rapport de prédictions positives avérées sur le total de prédictions positives émises par le modèle. La formule est comme suit :

$$P = \frac{VP}{VP + FP}$$

où VP est le nombre de vrais positifs et FP le nombre de faux positifs

Une précision qui s'éloigne de 1 (et donc s'approche de 0) indique que le modèle identifie plusieurs éléments comme positifs alors qu'ils sont négatifs. Donc plus la précision est proche de 1, plus le modèle est fiable lorsqu'il prétend qu'un élément est positif.

- *Rappel* : Cette métrique mesure le taux de détection des vrai positifs par le modèle :

$$R = \frac{VP}{VP + FN}$$

où FN est le nombre de faux négatifs

Un rappel qui s'éloigne de 1 (et s'approche de 0) indique que le modèle identifie plusieurs éléments comme négatifs alors qu'ils sont positifs. Donc plus le rappel est proche de 1, plus le modèle trouve les positifs (i.e. il manque peu de positifs ; i.e. les chances sont très faibles qu'un élément négatif soit en fait positif).

- *F1-score* : Cette métrique utilise les deux métriques précédentes comme suit :

$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Le F1-score est utile pour clarifier des situations contradictoires pour lesquelles, typiquement, la précision est élevée mais le rappel faible, ou inversement. Dans le cas où par exemple, la précision est élevée et le rappel faible, si on regarde que la précision, alors on conclut que le modèle est bon. Au contraire si on regarde que le rappel, alors on conclut que le modèle est mauvais, Ces conclusions peuvent être hâtives et le F1-score permet de trancher avec plus de raisons puisque ce score équilibre les deux métriques.

- *Matrice de confusion et extraction des métriques* : Cette matrice est une évaluation des décisions du modèle. Ce dernier établit pour chaque nouvel élément si celui-ci est positif ou négatif. Une fois que cette étape de décision est terminée (tous les éléments ont été classés entre positif et négatif), la matrice de confusion vient juger de ces décisions :

	Jugé positif	*Jugé négatif*
Réellement positif	VP	FN
Réellement négatif	FP	VN

VP : vrai positif ; FN : faux négatif ; FP : faux positif ; VN : vrai négatif

De cette matrice de confusion, il est possible de calculer la précision, le rappel, le F1-score. Cette matrice est donc un outil très utile postérieurement au triage de tous les éléments d'un échantillon.

- *Exactitude* :

$$E = \frac{VP + VN}{VP + FN + FP + VN}$$

Proportions de prédictions valides du modèle. Peut être calculée directement à partir de la matrice de confusion.

II. K plus proches voisins

- *Métrique de distance choisie et justification :*

La métrique de distance choisie est la distance euclidienne car cette métrique de distance est la plus utilisée pour les K plus proches voisins. De plus, dans les trois datasets utilisés, les données sont quantitatives et du même type, ce qui rend la distance euclidienne adaptée et un bon candidat pour la métrique de distance.

Ainsi pour tous vecteurs de caractéristiques x et y composés chacun de n

caractéristiques, nous avons :

$$D(x,y) = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i\right)^2}$$

- *Pseudo-code de l'algorithme :*

INIT(k_min, k_max, k_fold=10, **kwargs):

- Entrée :
  - k_min : Valeur minimale pour k (nombre de voisins).
  - k_max : Valeur maximale pour k.
  - k_fold : Nombre de partitions pour la validation croisée (par défaut 10).
  - kwargs : Autres paramètres optionnels.
- Action : Initialiser les attributs de la classe.
- Sortie : Aucune.

METHOD cross_validation(examples, labels):
- Entrée :
  - examples : Matrice (n_samples x n_features) des exemples.
  - labels : Vecteur (n_samples) des étiquettes.
- Action :
  1. Identifier les classes uniques.
  2. Parcourir les valeurs de k entre k_min et k_max.
  3. Pour chaque k, effectuer une validation croisée :
     - Diviser les données en k_fold groupes.
     - Calculer l'erreur moyenne sur les données de validation.
  4. Sélectionner la valeur de k ayant l'erreur moyenne minimale.
- Sortie : Met à jour `self.k` avec la meilleure valeur de k.

METHOD calculate_score_for_k(k, train_split, validation_split):
- Entrée :
  - k : Nombre de voisins à utiliser.

- train_split : Données d'entraînement (n_samples x n_features).
- validation_split : Données de validation (n_samples x n_features).

- Action : Évaluer la précision pour la valeur donnée de k.
- Sortie : Précision du modèle pour le k spécifié.

METHOD train(train, train_labels):

- Entrée :
  - train : Matrice (n_samples x n_features) des données d'entraînement.
  - train_labels : Vecteur (n_samples) des étiquettes associées.
- Action : Stocker les données d'entraînement et leurs étiquettes.
- Sortie : Aucune.

METHOD predict(x):

- Entrée :
  - x : Vecteur (1 x n_features) ou matrice (n_samples x n_features) des exemples à prédire.
- Action :
  1. Calculer les distances entre x et chaque exemple d'entraînement.
  2. Trouver les k voisins les plus proches.
  3. Déterminer la classe majoritaire parmi les voisins.
- Sortie : Classe prédite (ou liste de classes prédites si plusieurs exemples sont fournis).

METHOD evaluate(X, y):

- Entrée :
  - X : Matrice (n_samples x n_features) des exemples de test.
  - y : Vecteur (n_samples) des étiquettes réelles.
- Action :
  1. Prédire les classes pour chaque exemple de X.
  2. Construire une matrice de confusion.
  3. Calculer la précision globale.
  4. Calculer précision, rappel, et F1-score pour chaque classe.
- Sortie :
  - Dictionnaire contenant :

- "accuracy" : Taux de bonnes prédictions.

- "confusion_matrix" : Matrice de confusion.

- "metrics_per_class" : Dictionnaire des métriques pour chaque classe (précision, rappel, F1-score).

- *Évaluation sur données entrainement puis test :*

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.9778	0.8826	0.8587
Matrice de confusion		$\begin{bmatrix} 29 & 0 & 0 \\ 0 & 29 & 1 \\ 0 & 1 & 30 \end{bmatrix}$	$\begin{bmatrix} 917 & 94 \\ 96 & 512 \end{bmatrix}$	$\begin{bmatrix} 175 & 87 & 0 \\ 39 & 1887 & 27 \\ 0 & 201 & 89 \end{bmatrix}$
Précision	Classe 0	1	0.9	0.82
	Classe 1	0.967	0.84	0.87
	Classe 2	0.967	NA	0.77
Rappel	Classe 0	1	0.9	0.67
	Classe 1	0.967	0.84	0.97
	Classe 2	0.968	NA	0.31
F1-score	Classe 0	1	0.9	0.74
	Classe 1	0.967	0.84	0.91
	Classe 2	0.968	NA	0.44

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.9667	0.7806	0.8061
Matrice de confusion		$\begin{bmatrix} 21 & 0 & 0 \\ 0 & 18 & 2 \\ 0 & 0 & 19 \end{bmatrix}$	$\begin{bmatrix} 530 & 98 \\ 139 & 313 \end{bmatrix}$	$\begin{bmatrix} 101 & 85 & 0 \\ 51 & 1202 & 33 \\ 0 & 155 & 44 \end{bmatrix}$
Précision	Classe 0	1	0.79	0.66
	Classe 1	1	0.76	0.83
	Classe 2	0.9	NA	0.57
Rappel	Classe 0	1	0.84	0.54
	Classe 1	0.9	0.69	0.93
	Classe 2	1	NA	0.22
F1-score	Classe 0	1	0.82	0.60
	Classe 1	0.947	0.72	0.88
	Classe 2	0.95	NA	0.32

- *Résultats scikit-learn sur les données de test:*

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.95	0.7703	0.8043
Matrice de confusion		$\begin{bmatrix} 21 & 0 & 0 \\ 0 & 17 & 3 \\ 0 & 0 & 19 \end{bmatrix}$	$\begin{bmatrix} 523 & 105 \\ 143 & 309 \end{bmatrix}$	$\begin{bmatrix} 98 & 88 & 0 \\ 52 & 1190 & 44 \\ 0 & 143 & 156 \end{bmatrix}$
Précision	Classe 0	1	0.79	0.65
	Classe 1	1	0.75	0.84
	Classe 2	0.86	NA	0.56
Rappel	Classe 0	1	0.83	0.53
	Classe 1	0.85	0.68	0.93
	Classe 2	1	NA	0.28
F1-score	Classe 0	1	0.81	0.58
	Classe 1	0.92	0.71	0.88
	Classe 2	0.93	NA	0.37

III. Classification Naïve Bayésienne

- *Pseudo-code de l'algorithme :*

INIT:

- Entrée : Aucune.
- Action : Initialiser les variables suivantes :
  - classes : Liste des classes uniques dans les données.
  - means : Moyennes des features par classe.
  - stds : Écarts-types des features par classe.
  - priors : Probabilités a priori des classes.
- Sortie : Aucune.

METHOD train(train_data, train_labels):

- Entrée :
  - train_data : Matrice (n_samples x n_features) des exemples d'entraînement.
  - train_labels : Vecteur (n_samples) des étiquettes correspondantes.
- Action :
  1. Identifier les classes uniques dans train_labels.

2. Initialiser les tableaux `means`, `stds`, et `priors` pour chaque classe.

3. Pour chaque classe :

- Filtrer les exemples appartenant à cette classe.
- Calculer la moyenne et l'écart-type pour chaque feature.
- Calculer la probabilité a priori de cette classe.

4. Remplacer les écarts-types nuls par une petite valeur (ex. 1e-6).

- Sortie : Aucune (met à jour les variables internes de la classe).

METHOD predict(x):

- Entrée :
  - x : Vecteur (n_features) ou matrice (n_samples x n_features) des exemples à prédire.
- Action :
  1. Si x est un vecteur, le convertir en une matrice 2D.
  2. Initialiser une liste pour stocker les log-probabilités postérieures.
  3. Pour chaque classe :
     - Calculer le log de la probabilité a priori.
     - Calculer la somme des log-vraisemblances pour chaque feature.
     - Ajouter le log de la probabilité a priori aux log-vraisemblances.
  4. Identifier la classe ayant la probabilité postérieure maximale.
- Sortie :
  - Classe prédite (si un seul exemple est donné).
  - Liste des classes prédites (si plusieurs exemples sont donnés).

METHOD evaluate(X_test, y_test):

- Entrée :
  - X_test : Matrice (n_samples x n_features) des exemples de test.
  - y_test : Vecteur (n_samples) des étiquettes réelles.
- Action :
  1. Prédire les classes pour tous les exemples de X_test.
  2. Initialiser une matrice de confusion (n_classes x n_classes).

3. Remplir la matrice de confusion en comparant les prédictions avec y_test.

4. Calculer l'accuracy :

- Nombre de prédictions correctes / nombre total d'exemples.

5. Pour chaque classe :

- Calculer précision, rappel, et F1-score à partir de la matrice de confusion.

6. Stocker les métriques dans un dictionnaire.

- Sortie :

- Dictionnaire contenant :

- "accuracy" : Taux de bonnes prédictions.

- "confusion_matrix" : Matrice de confusion.

- "metrics_per_class" : Métriques pour chaque classe (précision, rappel, F1-score).

- *Résultats sur données entraînements puis tests :*

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.9444	0.7869	0.5804
Matrice de confusion		$\begin{bmatrix} 29 & 0 & 0 \\ 0 & 28 & 2 \\ 0 & 3 & 28 \end{bmatrix}$	$\begin{bmatrix} 777 & 234 \\ 111 & 497 \end{bmatrix}$	$\begin{bmatrix} 241 & 21 & 0 \\ 297 & 1062 & 594 \\ 4 & 135 & 151 \end{bmatrix}$
Précision	Classe 0	1	0.875	0.44
	Classe 1	0.90	0.68	0.87
	Classe 2	0.93	NA	0.20
Rappel	Classe 0	1	0.77	0.92
	Classe 1	0.93	0.82	0.54
	Classe 2	0.90	NA	0.52
F1-score	Classe 0	1	0.82	0.60
	Classe 1	0.918	0.74	0.67
	Classe 2	0.918	NA	0.29

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.9833	0.8037	0.5739
Matrice de confusion		$\begin{bmatrix} 21 & 0 & 0 \\ 0 & 19 & 1 \\ 0 & 0 & 19 \end{bmatrix}$	$\begin{bmatrix} 497 & 131 \\ 81 & 371 \end{bmatrix}$	$\begin{bmatrix} 169 & 16 & 1 \\ 222 & 686 & 378 \\ 1 & 94 & 104 \end{bmatrix}$
Précision	Classe 0	1	0.86	0.43
	Classe 1	1	0.74	0.86
	Classe 2	0.95	NA	0.22
Rappel	Classe 0	1	0.79	0.91
	Classe 1	0.95	0.82	0.53
	Classe 2	1	NA	0.52
F1-score	Classe 0	1	0.82	0.58
	Classe 1	0.974	0.78	0.66
	Classe 2	0.974	NA	0.30

- *Résultats scikit-learn sur les données de test :*

Dataset		Iris	Wine Quality	Abalones
Exactitude		0.95	0.7703	0.8043
Matrice de confusion		$\begin{bmatrix} 21 & 0 & 0 \\ 0 & 19 & 1 \\ 0 & 0 & 19 \end{bmatrix}$	$\begin{bmatrix} 498 & 130 \\ 80 & 372 \end{bmatrix}$	$\begin{bmatrix} 169 & 16 & 1 \\ 222 & 686 & 378 \\ 1 & 94 & 104 \end{bmatrix}$
Précision	Classe 0	1	0.86	0.65
	Classe 1	1	0.74	0.84
	Classe 2	0.95	NA	0.56
Rappel	Classe 0	1	0.80	0.53
	Classe 1	0.95	0.81	0.93
	Classe 2	1	NA	0.28
F1-score	Classe 0	1	0.83	0.58
	Classe 1	0.97	0.78	0.88
	Classe 2	0.97	NA	0.37

IV. Discussion des résultats

➢ *Classification K plus proches voisins :*

Les exactitudes du modèle Knn sur le corpus d'entraînement selon le dataset est 0.98, 0.88 et 0.86. Pour le corpus de test, les exactitudes sont de 0.97,0.78 et 0.80. Il peut être déduit que l'entraînement s'est très bien déroulé, et les tests également mais les deux exactitudes de 0.8 et 0.78 impliquent toutefois la présence moins négligeables de difficultés. Les scores F1 pour chaque classe de chaque dataset sont très élevés, donc viennent confirmer la réussite du modèle Knn avec les données. Néanmoins, les classes 0 et 2 du dataset d'Abalones ont posé des difficultés significatives au modèle (malgré une exactitude élevée) puisque les scores F1 d'entraînement et de tests sont faibles. Cela s'explique par la disproportion au niveau de la quantité de données entre la classe 1 et les deux autres classes; cette information est donnée par la matrice de confusion, puisqu'en entraînement, la classe 0 comporte 262 échantillons, la classe 1 en a 1953 et la classe 2 en a 290. La disproportion est clairement visible, et elle se propage sur les données de test. Pour le dataset d'Iris, la proportion pour chaque classe est presque égale, c'est pourquoi les résultats sont les meilleurs, malgré une faible quantité de données. Le dataset de Wine Quality a une proportion d'environ 60-40 entre ses deux classes, donc une proportion acceptable, et la classification est binaire donc il y

a possiblement moins d'ambiguïté débouchant sur de meilleurs résultats que s'il y avait plus de classes.

Précédemment, il a été mentionné que les exactitudes du modèle étaient légèrement inférieures sur les données de test que les données d'entraînement. De manière générale, cette tendance est confirmée par les mesures de précision, rappel et score-f1; pour chaque cellule dans le tableau d'entrainement, la cellule correspondante dans le tableau de test contient une valeur égale ou inférieure.

Pour formuler une conclusion sur les forces et les faiblesses de notre implémentation du modèle Knn, ce dernier réussit convenablement ses prédictions autant sur un dataset léger en nombre de données que sur un dataset fourni. Il est également performant sur une classification binaire ou ternaire, mais nécessite une proportion similaire de données entre les différentes classes. Dans le cas contraire, il ne parvient pas à prédire les échantillons de classes faiblement représentées.

Les résultats du modèle de la librairie Scikit Learn montrent que ce dernier a réussi de manière très équivalente à notre modèle aux mêmes endroits, c'est-à-dire pour les datasets Iris et Wine Quality. Pour le dataset problématique Abalones, le modèle Scikit learn a fourni une performance similaire, sinon mieux sur les classes faiblement représentées (0.58 au lieu de 0.6 de notre modèle pour la classe 0, et 0.37 au lieu de 0.32 pour la classe 2). Globalement, notre implémentation est très fidèle au modèle théorique puisque nos résultats sont presque égaux aux résultats du standard (i.e le modèle de Scikit).

- *Classification Naïve Bayésienne :*

Les exactitudes du modèle Naïf Bayésien sont de 0.94, 0.79 et 0.58 pour les corpus d'entraînement des trois datasets, et 0.98, 0.80 et 0.57 pour les corpus de tests. Contrairement au modèle Knn, les exactitudes calculées par le modèle montrent immédiatement les difficultés du modèle en présence de disproportion de distribution entre les classes.

Pour les exactitudes énumérées ci-dessus, et pour les autres mesures (précision, rappel et score-f1), il est curieux de constater que notre implémentation du modèle Naïf Bayésien réussit mieux en phase de test qu'en phase

d'entraînement. Ce prédicat provient de la comparaison de chaque cellule correspondante entre les deux tableaux (entraînement et test). Cette propriété est inverse au modèle Knn qui réussit mieux en phase d'entraînement que dans sa phase de test. Cette contradiction repose sur la différence fondamentale dans la conception de ces deux modèles, puisque le modèle Knn met l'accent sur sa méthode d'entraînement (avec la validation croisée) tandis que le modèle Naïf Bayésien a une méthode de test plus performante que sa méthode d'entraînement assez classique.

Les conclusions que l'on peut émettre sur les faiblesses/force de notre implémentation sont assez similaires que celles postulées pour le modèle Knn; notre implémentation du modèle Naïf Bayésien est performante sur des datasets composés de classes binaires/ternaires, que ces datasets soient légers ou fournis. On peut toutefois mentionner l'importance particulière de la distribution des échantillons à travers les classes, car la disproportion des données au dataset Abalones a infligé de sérieuses difficultés à notre modèle (les scores-F1 entraînement sont de 0.6, 0.67 et 0.29 pour les trois classes, et les scores-F1 de test sont 0.58, 0.66 et 0.3, ce qui est globalement très faible).

Les résultats du modèle Naïf Bayésien de Scikit Learn montrent que la faiblesse évidente de notre modèle en situation de mauvaise distribution n'est pas causée par une faiblesse d'implémentation, puisque les score-F1 du modèle Scikit sont de 0.58, 0.88 et 0.37 sur le corpus de test. L'exactitude est toutefois bien plus élevée mais nous avons vu qu'avec le modèle Knn qu'une exactitude élevée ne propose pas nécessairement une représentation fidèle du comportement du modèle. Ainsi, en comparant notre implémentation avec le modèle Scikit, l'addition de la réussite égale sur les datasets Iris et Wine Quality avec la similarité des scores-F1 faibles sur le dataset Abalones nous affirment que notre implémentation est assez fiable. Le modèle Naïf Bayésien de Scikit a possiblement une optimisation propre pour faire face aux mauvaises distribution et qui expliquerait les résultats légèrement meilleurs pour le dataset d'Abalones, mais globalement notre implémentation peut être qualifiée d'implémentation équivalente fiable.

V. Comparaison des techniques

Voici les moyennes calculées sur les trois datasets pour comparer les performances globales des deux approches :

	K Plus proches voisins (KNN)	*Classification Naïve Bayésienne*
Temps d'exécution	1,88s	0,75s
Accuracy	0,85	0,78
Précision	0,81	0,76
Recall	0,77	0,82

Les deux algorithmes présentent des performances relativement proches. KNN montre une meilleure accuracy et précision, tandis que Naive Bayes obtient un meilleur recall. Cela montre que KNN est plus précis dans ses prédictions, et que Naive Bayes est meilleur pour détecter les cas positifs. Le temps d'exécution moyen montre que Naive Bayes est environ 2,5 fois plus rapide que KNN. Cette différence s'explique par la complexité algorithmique : KNN doit calculer les distances avec tous les points à chaque prédiction, tandis que Naive Bayes utilise simplement des probabilités pré-calculées.

VI. Conclusion

Les résultats montrent des performances variables selon la complexité des datasets. Le jeu de données Iris, avec ses classes biens séparées, obtient d'excellentes performances avec les deux méthodes. Le dataset Wine présente de bons résultats malgré la complexité des données chimiques. Le cas des Abalones s'avère plus difficile, avec des performances modérées dues au chevauchement des classes d'âge.

L'implémentation a présenté plusieurs défis techniques. Dans l'implémentation de Naive Bayes, les très petites probabilités causaient des erreurs

de calcul, ce qui a été résolu en utilisant des logarithmes plutôt que les probabilités directes. Nous avons également dû gérer les écarts-types nuls qui causaient des divisions par zéro, en ajoutant epsilon dans ces cas. La conversion des données catégorielles en valeurs numériques a aussi nécessité une attention particulière. Dans l'implémentation des K plus proches voisins, pendant l'étape de validation croisée, le nombre d'exemples utilisés n'est pas toujours divisible par le nombre de plis que l'on souhaite créer (k_fold). Il a donc fallu prendre en compte les cas où les plis ont un nombre différent d'exemples lors de la répartition des plis dans les ensembles d'entraînement et de validation dans cette étape.

La comparaison entre les deux méthodes montrent que KNN offre une approche plus intuitive mais demande plus de temps de calcul. Naive Bayes se montre plus rapide mais repose sur des hypothèses plus strictes. Les résultats obtenus, très proches de ceux de scikit-learn, valident nos implémentations. Ces deux approches se complètent bien, chacune présentant ses avantages selon le type de données à traiter.