Rapport de projet

Projet de Classification

Fréjoux Gaëtan & Niord Mathieu

frejoux.gaetan@etu.univ-poitiers.fr
mathieu.niord@etu.univ-poitiers.fr

Université de Poitiers,
Master 2 Informatique - Conception Logicelle

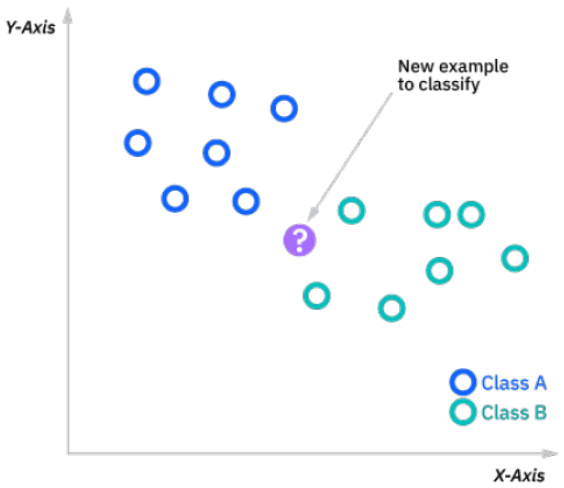
5 décembre 2022

Table des matières

1 Algorithme

Tout d'abord, contextualisons ce à quoi correspond l'algorithme du **K-ppv** (K plus proches voisins), plus connu sous son appellation anglaise ***Knn*** pour ***K-nearest neighbors***. Le *K-ppv* est un algorithme **non-paramétrique d'apprentissage supervisé** qui a pour but de déduire (et non de déterminer puisque faillible) la classification d'une donnée inconnue, en mesurant statistiquement la plus haute correspondance de classe parmi les données d'apprentissage qui l'avoisinent. Voici les étapes de déroulement de l'algorithme du **Kppv**.

[**Étape 1**] Pour ***k*** voisins parmi notre ensemble d'apprentissage ***apprent***, nous souhaitons classifier une donnée ***data***. Partons du principe que ces données sont inscrites dans un modèle de coordonnées à 2-dimensions (i.e. couple x, y) et qu'il existe uniquement 2 niveaux de classification, la **classe A** puis la **classe B**.



[**Étape 2**] Au cours de son exécution, l'algorithme détermine les ***k*** plus proches voisins de ***data*** parmi l'ensemble des données connues de ***apprent***.

[**Étape 3**] Une fois que l'algorithme a déterminé les k plus proches voisins de notre donnée, il recherche la plus grande occurrence de classification parmi notre nouvel ensemble et retourne la classification estimée.

1.1 Donnez quelques éléments de commentaires sur la stratégie que vous avez utilisée pour réaliser l'algorithme KppV.

Afin de réaliser notre Kppv "maison", nous devions d'abord récupérer nos données **d'apprentissage** et de **test** (à savoir, nos individus non classifiés). Le professeur nous a fourni les classifications de nos données d'apprentissage et le nombre de voisins (i.e. k) est à généraliser puisque sa valeur est laissée au libre choix de l'utilisateur. Voici en guise d'illustration la signature de notre fonction : `def kppv(apprent, classe_origine, k, X)`, où :

- `apprent` correspond à notre ensemble d'apprentissage,
- `classe_origine` aux classifications de ces dernières,
- `k` au nombre de voisins,
- `X` à notre ensemble de données à classifier.

Les librairies qui nous ont été utiles à l'élaboration de cet algorithme sont `numpy` et `scipy`.

Passons à l'algorithme. Une boucle parcourant l'ensemble des données de la matrice `X`. Pour chacune de ces données, l'algorithme récupère les paires de coordonnées puis initialise un tableau à l'intérieur duquel sont stockées les distances de l'individu par rapport aux données d'apprentissage. Pour cela, nous avons utilisé la fonction `linalg.norm()` du package `numpy` comme présenté ci-dessous.

```
data_to_classify = X[:, i]  # Récupération des coordonnées x et y
# Création du tableau des distances
distances = np.linalg.norm(apprentT - data_to_classify, axis=1)
```

L'algorithme se poursuit en récupérant les placements des k plus proches valeurs à partir desquels il peut récupérer la liste des classifications associées.

```
# Récupération des identifiants de nos k voisins dans un tableau
nearest_neighbor_ids = distances.argsort()[:k]
nearest_neighbor_classes = [
    classe_origine[id] for id in nearest_neighbor_ids
] # Mapping qui traduit les identifiants en classifications
```

La boucle se termine par une réduction statistique du tableau des classifications de nos plus proches voisins, puis le résultat est ajouté en queue d'un tableau dédié dans lequel sont stockés toutes les estimations de classification qui viennent d'être réalisées.

```
res.append(stats.mode(nearest_neighbor_classes, keepdims=True)[0][0])
```

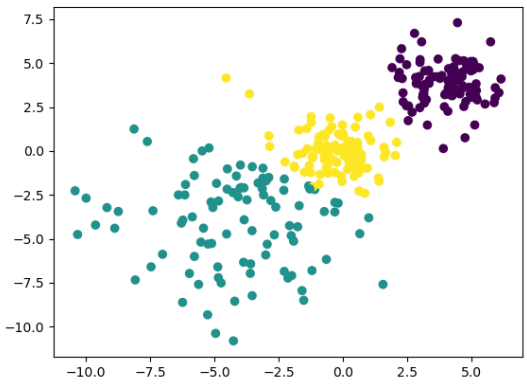
La fonction retourne le tableau comprenant les déductions de classification de nos données de test.

1.2 Analysez le comportement de votre algorithme face à l'ensemble de test et aux réglages des hyper-paramètres.

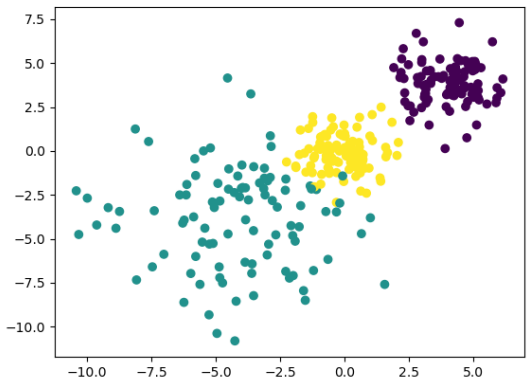
Le sujet nous demandait d'exécuter l'algorithme du ***Kppv*** sur une matrice d'apprentissage de 150 individus, une matrice d'individus à classifier et avec $k \in [1, 3, 5, 7, 13, 15]$. Voici un tableau regroupant le pourcentage de taux d'erreur en fonction de la valeur de k :

TABLE 1 – K plus proches voisins

Valeur de k	1	3	5	7	13	15
Taux d'erreur (en %)	6.0	4.34	3.34	2.67	2.0	2.67



Kppv



Résultats attendus

2 Tests de discrimination sur deux ensembles d'apprentissage associés aux mêmes phénomènes, l'un de grande taille, l'autre de petite taille

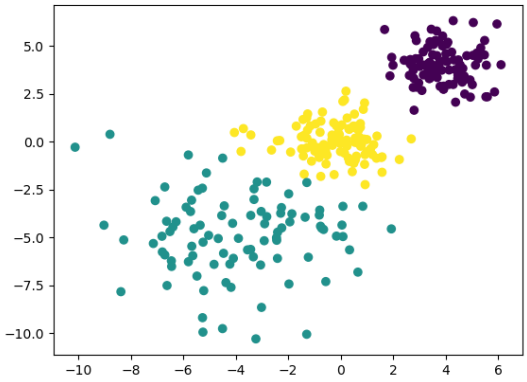
Dans cette partie, nous appliquons les algorithmes de classification du Knn (voir ci-dessus) et du Bayes Naïf sur un ensemble de 300 individus inconnus et deux ensembles d'apprentissage. Notre premier ensemble (p1_petit.xlsx) contient 60 individus divisés en trois sections de 20 individus appartenant respectivement à la classe 0, la classe 1 et la classe 2. La répartition des individus est identique sur notre ensemble (p1_grand.xlsx) mais cette fois-ci sur 450 individus, soit trois sections de 150 individus.

2.1 Pour chacun, indiquez le taux d'erreur sous forme d'un tableau. Vous pouvez étudier visuellement dans certains cas où sont les erreurs et commenter

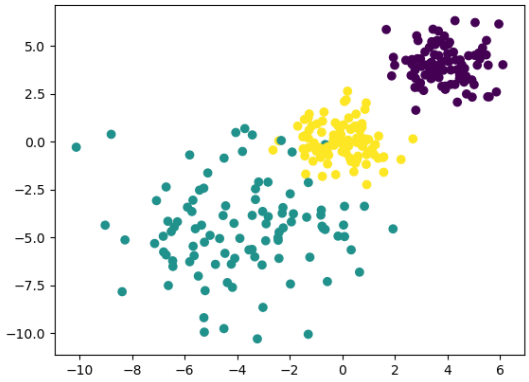
TABLE 2 – Taux d'erreur du knn en fonction de k sur un petit ensemble

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	3.67	3.0	2.33	2.67	3.67	3.67

Sur ce premier ensemble, établi tel que "petit" du haut de ses 60 individus, nous observons un taux d'erreur de l'ordre de 2.33% dans le meilleur des cas au sein de notre échantillon de tests, lorsque la valeur de k est égale à 5.



Knn sur petit ensemble



Résultats attendus

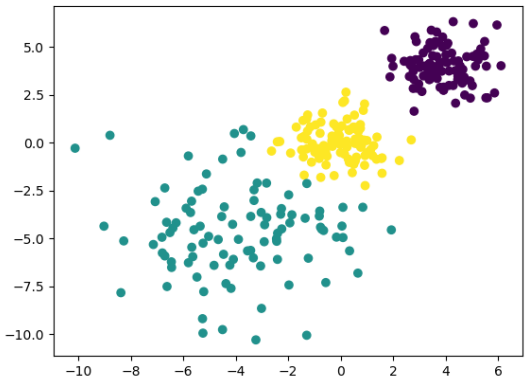
En comparant visuellement notre meilleure répartition à la répartition attendue, nous remarquons que nos erreurs se concentrent principalement si ce n'est qu'en intégralité au centre de notre graphique, sur la frontière définie entre notre classe 0 (en bleu) et notre classe 1 (en jaune). On pourrait penser que pour palier à ce problème il nous suffirait de travailler sur davantage de données afin de mieux définir notre frontière.

Quand nous effectuons ce même traitement sur un ensemble de 450 individus, l'échantillon étant découpé de manière à ce que les 150 premiers soient de la classe 0, les 150 suivants de la classe 1 et enfin les 150 derniers soient de la classe 2, voici les résultats que nous obtenons :

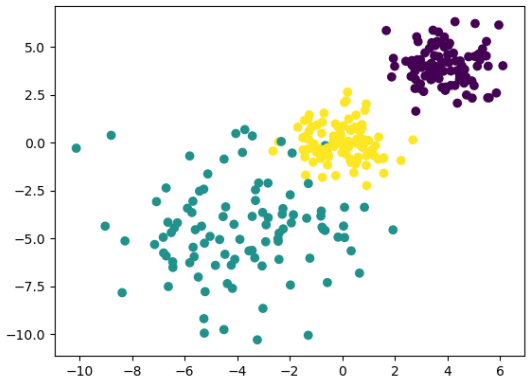
TABLE 3 – Taux d'erreur du knn en fonction de k sur un grand ensemble

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	2.0	1.67	1.33	1.0	1.67	1.33

Cette fois ci nous observons un taux d'erreur relativement plus bas. En comparant notre pire cas (ici pour k = 1) nous remarquons qu'il est déjà moins élevé que le meilleur cas observé ci-dessus (2.0% ∱ 2.33%). Le taux d'erreur le plus bas observé correspond à k = 7 avec seulement 1% d'erreurs. De manière plus générale, on peut penser que la quantité de données joue un rôle pondérant quant à la petitesse de nos taux d'erreur.



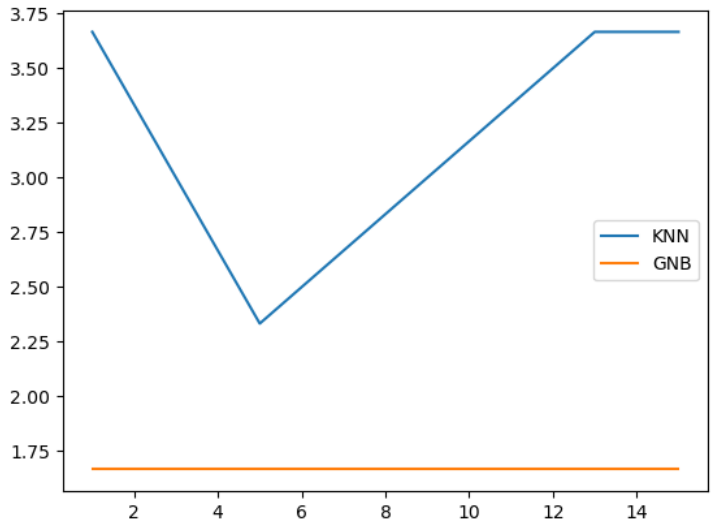
Knn sur grand ensemble



Résultats attendus

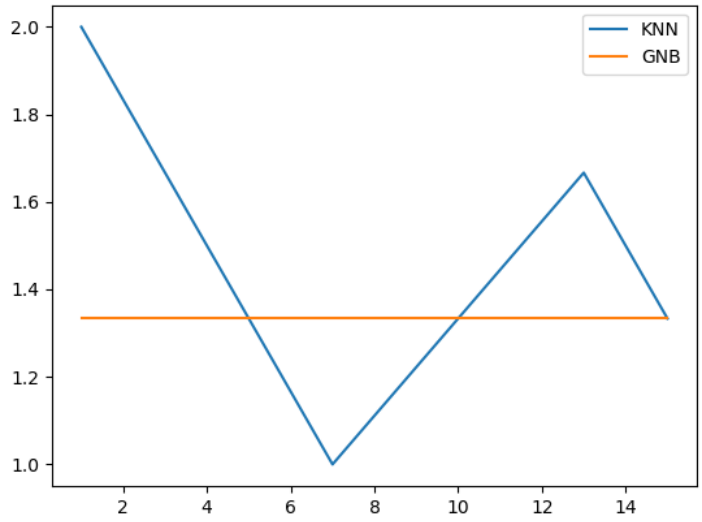
La différence est plus subtile à observer puisqu'elle ne concerne finalement qu'environ 4,5 individus. On pourra toutefois remarquer que les individus concernés se trouvent en plein dans une zone de classe 1 (en jaune) quand ils sont censés faire partie de la classe 0 (en bleu). Afin de palier à cette incongruité, il faudrait pouvoir soit mieux définir les ensembles/propriétés qui définissent nos catégories, soit augmenter le nombre de voisins au grand risque d'obtenir un taux d'erreur nettement supérieur à celui-ci. Il est également très probable que la répartition proposée (donc attendue par le professeur) soit biaisée car trop linéaire (rappelons-le, nous divisons notre ensemble d'apprentissage en trois sections sans qu'il n'y est de pré-classement des individus) d'où le chevauchement de certaines familles.

2.2 Étudiez l'évolution de l'erreur en fonction de K, et la différence avec Bayes naïf. Expliquez cette évolution, et la comparaison avec Bayes.



Différences du taux d'erreur entre Knn et Bayes Naïf sur un petit ensemble d'individus

Rien à dire ici, le graphique parle de lui-même. L'algorithme de **Bayes Naïf** est nettement plus efficace que l'algorithme **Knn**, et ce pour toutes les valeurs de $\boldsymbol{k}$. Toutefois comment expliquer l'évolution de l'erreur en fonction de la valeur de $\boldsymbol{k}$? Comme nous avons pu le voir un peu plus haut sur les figures comparatives, les erreurs se concentrent à la frontière entre les classes **0** et **1**. Par ailleurs, nous pouvons observer que la répartition des individus de la classe 0 (en bleu) est bien plus étendue que celle des classes **1** et **2** qui forment des ensembles bien mieux définis. En ajoutant à cela que l'ensemble d'apprentissage n'est pas assez développé pour pouvoir combler ce manque de clarté.



Différences du taux d'erreur entre Knn et Bayes Naïf sur un grand ensemble d'individus

Cette fois, nous observons quelque chose d'intéressant. En effet, on observe que l'algorithme **Knn** quand $\boldsymbol{k}$ est égal à 7 est meilleur que l'algorithme de **Bayes Naïf**.

2.3 Comparez les deux ensembles d'apprentissage en mettant en évidence les différences (ou non) pour les deux méthodes. Expliquez pourquoi ces différences selon vous.

On observe que de manière générale, l'algorithme de **Bayes Naïf** est plus performant que l'algorithme **Knn**. Toutefois, l'algorithme **Knn** peut être plus performant que l'algorithme de **Bayes Naïf** dans certains cas.

3 Tests de discrimination sur un ensemble pré-classé par Coalescence. Cet ensemble présente probablement des erreurs

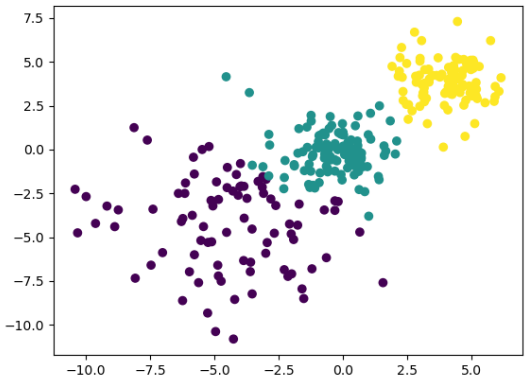
Dans cette partie, nous appliquons les algorithmes de classification du **Knn** (voir ci-dessus) et du **Bayes Naïf** sur un ensemble de **300 individus inconnus** et un ensemble d'apprentissage. L'ensemble d'apprentissage contient **150 individus classés** par un **algorithme de clustering** (***Kmeans***) par absence de professeur. Par conséquent, l'ensemble d'apprentissage présente (certainement) des erreurs.

3.1 Indiquez le taux d'erreur sous forme d'un tableau. Vous pouvez étudier visuellement dans certains cas où sont les erreurs et commenter

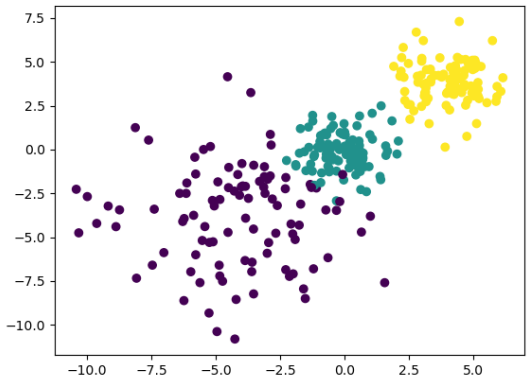
TABLE 4 – Taux d'erreur en fonction de k avec un ensemble pré-classé par un algorithme de clustering (K-Means)

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	6.67	6.0	5.0	5.0	5.33	5.67

Nous observons un meilleur cas pour deux valeurs de ***k*** : **5** et **7**. Pour ces deux valeurs nous obtenons le taux d'erreur le plus bas de notre échantillon de tests, c'est à dire 5,0%, ce qui est bien plus élevé que ce nous avons pu voir auparavant.



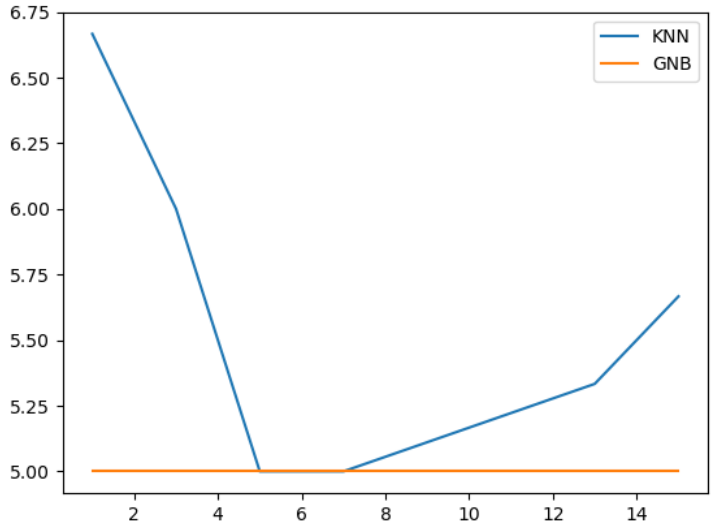
Knn sur un ensemble pré-classé



Résultats attendus

Graphiquement, on observe que les différences entre notre résultat et celui espéré, se concentrent, encore, exclusivement au niveau de la frontière entre nos classes 0 et 1.

3.2 Étudiez l'évolution de l'erreur en fonction de K, et la différence avec Bayes naïf. Expliquez cette évolution, et la comparaison avec Bayes.



Différences du taux d'erreur entre Knn et Bayes Naïf sur un ensemble pré-classé d'individus

L'algorithme de **Bayes Naïf** est toujours plus performant que l'algorithme **Knn**. Le taux d'erreur de l'algorithme de **Bayes Naïf** est égal à 5%. Pour l'algorithme **Knn**, le taux d'erreur minimum est égal à 5% quand ***k*** est égal à 5 et 7. On a donc ici, pour ***k*** égal à 5 et 7, un taux d'erreur égal à celui de l'algorithme de **Bayes Naïf**. Cependant, pour les autres valeurs de ***k***, le taux d'erreur est supérieur à celui de l'algorithme de **Bayes Naïf**.

3.3 Étudiez les différences de tendance avec le test précédent. Expliquez les différences de comportement.

Tout d'abord, on observe que le taux d'erreur est beaucoup plus élevé que sur l'ensemble précèdent (environ 3 fois plus élevé). On observe que l'algorithme de **Bayes Naïf** est toujours plus performant que l'algorithme **Knn** de manière générale. Le taux d'erreur de l'algorithme **Knn** évolue, dans les grandes lignes, de la même manière que le test précèdent et le meilleur choix de ***k*** reste entre 5 et 7.

4 Tests de discrimination sur un ensemble ne suivant pas une distribution Gaussienne

Dans cette partie, nous appliquons nos algorithmes sur une distribution non gaussienne. l'ensemble d'apprentissage contient **210 individus**.

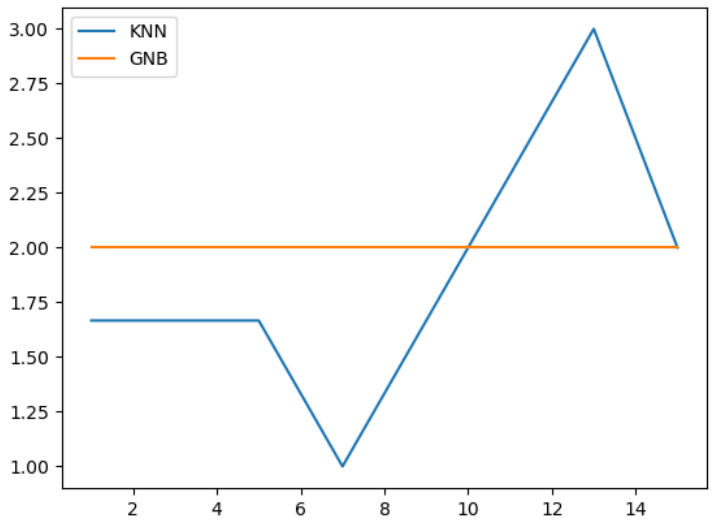
4.1 Indiquez le taux d'erreur sous forme d'un tableau. Vous pouvez étudier visuellement dans certains cas où sont les erreurs et commenter

TABLE 5 – Non Gaussien

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	1.67	1.67	1.67	1.0	3.0	2.0

On observe pour une distribution non gaussienne un taux d'erreur plutôt bas avec un pire taux de 3% quand ***k*** est égal à 13 et un meilleur taux de 1% quand ***k*** est égal à 7.

4.2 Étudiez l'évolution de l'erreur en fonction de K, et la différence avec Bayes naïf. Expliquez cette évolution, et la comparaison avec Bayes.



Différences du taux d'erreur entre Knn et Bayes Naïf sur un ensemble non-gaussien d'individus

On observe que l'algorithme **Knn** est très performant sur cette distribution non-gaussienne. En effet, le taux d'erreur minimum est égal à 1% quand $\boldsymbol{k}$ est égal à 7. L'algorithme de **Bayes Naïf** quant à lui, est moins performant avec un taux d'erreur de 2%. Qui plus est, on observe que l'algorithme **Knn** est toujours plus performant que l'algorithme de **Bayes Naïf** quand $\boldsymbol{k}$ vaut 1, 3, 5 et 7.

4.3 Êtes-vous surpris par les résultats ? Expliquez pourquoi.

Avant toute chose, les résultats sont pour la première fois en faveur de l'algorithme **Knn** sur l'ensemble de test. Ensuite on observe malgré tout que l'algorithme de **Bayes Naïf** est toujours plus performant que l'algorithme **Knn** quand $\boldsymbol{k}$ est égal à 13 et 15. Et ce, même si nous utilisons une distribution non gaussienne alors même que l'algorithme est basé sur une distribution gaussienne.

5 Bilan

Faites un bilan sur le comportement des deux méthodes en justifiant les différences à partir de leur fonctionnement (paramètres, types de frontières) ...

En résumé, on observe que le taux d'erreurs est plus bas sur les grands ensembles que sur les petits ensembles. Nous avons également pu remarquer que la labélisation des données d'apprentissage permet d'obtenir des résultats plus précis. Globalement, l'algorithme de **Bayes Naïf** Gaussien est plus performant que l'algorithme **Knn** sur les ensembles de test suivant une distribution gaussienne et qu'à l'inverse, l'algorithme **Knn** est plus performant que l'algorithme **Bayes Naïf** sur les ensembles de test ne suivant pas une distribution gaussienne.

6 Volet Chargement des descripteurs

Explicitez comment sous Python vous avez importé et créé votre corpus : vecteur de labels, ensemble d'apprentissage, ensemble de tests.

Parce que rien ne vaut une bonne conception pour mieux s'y retrouver, nous avons adopter les préceptes de la clean architecture dans un premier temps. Plus exactement, nos fichiers de données brutes (à savoir les fichiers fournis pour les besoins de ce projet)

sont tous stockés à l'intérieur d'un répertoire de ressources judicieusement nommé *res* que vous pourrez trouver à la racine de notre projet, à côté d'un second répertoire *src* à l'intérieur duquel se trouve notre code source.

Pour encore mieux s'organiser, nous avons choisi de regrouper l'ensemble des liens vers nos fichiers ressources à l'intérieur d'un fichier ressources.py sous la forme de constantes. C'est d'ailleurs à l'intérieur de ce même fichier que l'utilisateur aura la possibilité de modifier sa configuration concernant les valeurs de $\boldsymbol{k}$ testées ou bien encore la sauvegarde des figures (ou graphiques) générées à l'aide de la librairie `matplotlib`. Cela étant fixé, nous pouvons passer au vif du sujet, soit le chargement de nos descripteurs dans la partie 3 du sujet **Classification**.

Pour importer notre corpus depuis le tableur *./res/WangSignature.xls*, nous nous sommes servis de la librairie pandas qui fourni une méthode **read_excel()** appropriée à notre besoin. Pour chacun de nos descripteurs, à savoir *PHOG*, *JCD*, *CEDD*, *FCTH* et *FuzzyColorHistogr*, nous avons créer un vecteur de labels *CLASSIF_WANG* que nous avons rempli à l'aide d'une boucle. Chaque valeurs ajoutée à *CLASSIF_WANG* correspond à la classe d'une image, déterminée à partir de son nom (dans notre cas la valeur de l'itération suivi de l'xetension "*.jpg*") convertit en entier puis divisé par 100. Le tableau de classification résultant nous apprend que toutes les images comprises entre 0 et 99 sont de classe **0**, de 100 à 199 de classe **1**, de 200 à 299 de classe **2**, etc... jusqu'à la 999ème image.

Afin de construire notre corpus, il a fallu définir notre ensemble de tests et notre ensemble d'apprentissage. Pour se faire, nous sommes passés par la méthode `train_test_split()` de la librairie `sklearn`. Cette méthode nous a permis de générer deux ensembles (*train* et *test*) par distribution aléatoire des données qui, par choix empirique, représentent respectivement 80% et 20% de notre ensemble de départ.

7 Classification

Toutes nos valeurs sont obtenues avec l'utilisation de la méthode **train_test_split** avec pour ***random_state*** la valeur **100**

7.1 Comparez la qualité de la discrimination en fonction de la caractéristique utilisée ou du cumul de toutes les mesures

On observe que le taux d'erreur est élevé, peu importe le descripteur utilisé. Le **CEDD** est la meilleur solution de description dans notre cas ainsi que le cumul de tous les descripteur avec un taux d'erreur égal à ***0.225*** (voir tableau ci-dessous)

TABLE 6 – Résultat Bayes en fonction des descripteurs

PHOG	0.425
JCD	0.235
CEDD	0.225
FCTH	0.230
FCH	0.385
ALL	0.225

7.2 Avec les meilleures combinaisons de mesures, paramétrez au mieux l'algorithme KppV

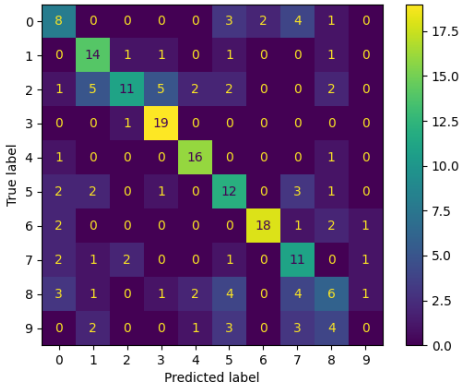
Nous constatons (voir figures ci-dessous) que l'algorithme KppV dans notre cas est plus performant avec des valeurs de **k** assez grande. En effet, contrairement aux parties précédentes, on observe les taux d'erreurs les plus bas quand **k** vaux **7**, **13** ou **15**. Par exemple, pour **JCD**, on obtient le taux d'erreur le plus faible lorsque **k** vaut 13

7.3 Avec la meilleure configuration de K, comparez KppV et Bayes

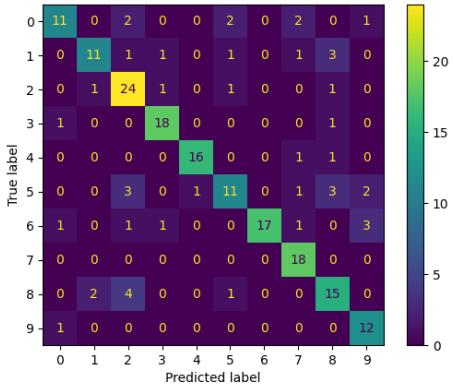
On observe que l'algorithme KppV est dans notre cas toujours plus performants que l'algorithme Bayes. Par exemple, nous avons avec **k** égal à **13** un taux d'erreur à **0.14** avec le descripteur JCD alors que le taux d'erreur obtenu est de **0.235** pour Bayes.

7.4 Avec le meilleur discriminateur évaluez la qualité de la procédure de discrimination selon la classe d'images.

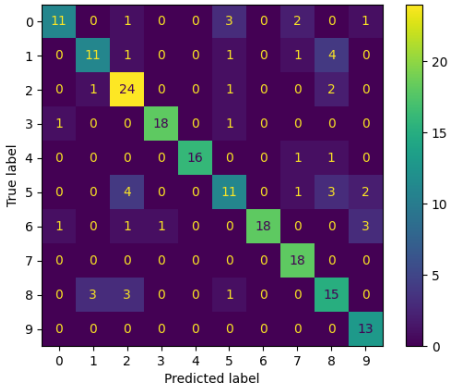
Dans notre cas, le meilleur discriminateur est **CEDD**. On observe à l'aide de la matrice de confusion que la classe 3 est la classe la mieux déterminée et les classes 0, 1 et 5 celles qui le sont le moins.(voir les matrices de confusion ci-dessous).



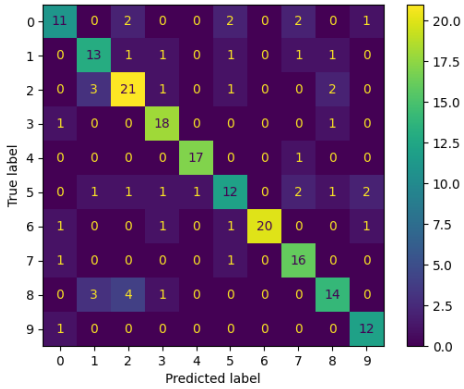
(a) PHOG_CM



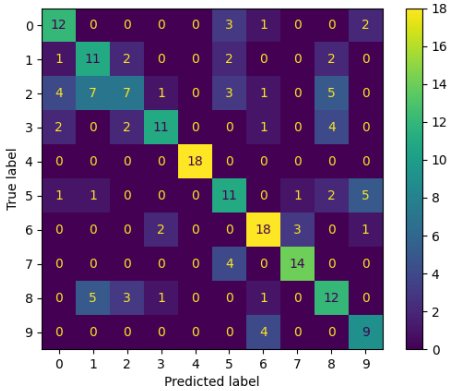
(b) JCD_CM



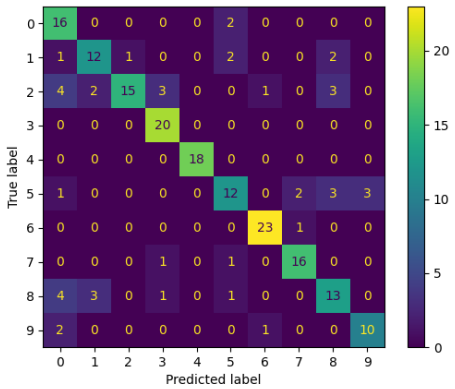
(c) CEDD_CM



(d) FCTH_CM



(e) FCH_CM



(f) ALL_CM

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.465	0.445	0.440	0.440	0.415	0.435

(a) KNN_PHOG

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.210	0.190	0.185	0.175	0.140	0.145

(b) KNN_JCD

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.200	0.175	0.180	0.165	0.165	0.170

(c) KNN_CEDD

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.235	0.185	0.210	0.185	0.190	0.150

(d) KNN_FCTH

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.375	0.365	0.330	0.320	0.320	0.325

(e) KNN_FCH

Valeur de *k*	1	3	5	7	13	15
Taux d'erreur (en %)	0.340	0.305	0.300	0.275	0.295	0.290

(f) KNN_ALL