# Résultats de recherche sur la classification supervisée

M. Deker Sylvain[1] Mme. Courdy-Bahsoun Clémence[2]

[1] M1 IGAI Université Paul Sabatier
[2] M1 IGAI Université Paul Sabatier

sylvain.deker@univ-tlse3.fr
clemence.courdy-bahsoun@univ-tlse3.fr

## Résumé

*Cet article vise à comparer les avantages et les désavantages des classifications supervisées par bayésienne avec gaussienne et par la méthode des k-plus proches voisins, en s'appuyant sur les résultats obtenus sur un même jeu de données.*

## Mots Clef

Apprentissage, classification bayésienne, classification k-ppv.

# 1 Introduction

La classification bayésienne avec gaussienne et l'approche par la méthode des k-ppv sont des classifications supervisées dont l'objectif est de répartir les données par classe avec un taux d'erreur minimisé. Les tests sont menés sur un jeu de données contenant une centaine d'occurences de la prononciation d'une dizaine de voyelle aau format csv.
A l'aide des travaux réalisés précédamment et en adaptant un peu le code nous avons pu implémenter le modèle de Bayes. L'implémentation pour une approche par les k-ppv, le sujet nous imposait l'utilisation de *Scikitlearn*.
Dans les tests menés les données fournies ont été repartie de telle sorte à avoir 80% de données d'apprentissage et 20% de données de tests. Dans les démarches d'apprentissage le choix d'un taux de données d'apprentissage élevé favorise l'obtention de meilleurs résultats, ici une meilleur prédiction de la répartition par classe.

# 2 Classification bayésienne avec Gaussienne

La classification bayéssienne avec Gaussienne est une minimisation de la vraisemblance qui vise à prédire la classe d'appartenance la plus probable d'une donnée. Néanmoins avec des données réelles il est difficile d'obtenir des résultats satisfaisants car les données ne sont pas simples. Ainsi pour améliorer le modèle on tient compte de la probabilité a priori en applicant la loi de Bayes.
Dans les tps précedant, nous avions déjà implémenté en python l'algorithme de prédiction permettant de classer les données fournies. Ici nous somme reparti de ce code qui était basé sur une simplification de la vraisemblance en log-vraisemblance. C'est-à-dire que la vraisemblance qui est une expression exponentielle a été simplifié par l'application du logarithme népérien afin d'obtenir un calcul un peu plus optimisé. De plus toutes les constantes ne dépendant pas des données ont été supprimé, n'apportant pas une précision indispensable dans la reconnaissance des données.

# 3 Classification k-NN

La méthode des K plus proches voisins (K-NN) a pour but de classifier des points cibles (classe méconnue) en fonction de leurs distances par rapport à des points constituant un échantillon dápprentissage (cést-à-dire dont la classe est connue a priori). Le calcul de la distance peut dépendre de l'expérience menée, Scikitlearn se base sur un calcul euclidien de la distance entre les données de l'échantillon, placées dans un repère après un traitement par une **A**nalyse en **C**omposantes **P**rincipales, et les centroïdes de la base d'apprentissage. Scikit learn est une bibliothèque libre pour python dont certaines fonctionnalités ont été utilisé pour implémenter la classification par k-NN. En effet toutes les fonctions appelé sur la variable n de la fonction k_NN sont des fonctions appelés pour la classe KNeighborsClassifier, une classe de la bibliothèque permettant d'effectuer les traitements nécéssaire à une classification par plus proche voisin.

**Algorithm 1** k_NN

**Require:** train_data, train_labels, test_data, test_labels
**Ensure:** m : matrice de confusion
p : precision
n = neighbors.KNeighborsClassifier(n_neighbours,weights = 'distance')
n.fit(train_data,train_labels) %apprentissage
y = n.predict(test_data) mconf = matriceConf(test_labels,y) %calcul matrice de confusion, fonction rédigé
**return** (mconf[0], mconf[1])

# 4 Evaluation par validation croisée

La validation croisée est une méthode déstimation de la fiabilité dún modèle fondé sur la technique d'échantillonage. Les essembles de données d'apprentissages sont les fichiers de test data2.csv, data3.csv et data12.csv sur lequel on peut entrainer le modèle. Nous appelons "leanring ratio" la proportion des données d'un fichier dédié à lápprentissage, la partie restante sert déchantillons da validation. Cette méthode de test s'appelle "testset validation" ou "holdout method".

# 5 Résultats

Nous rappelons que le "leanring ratio" est la proportion des données d'un fichier dédié à lápprentissage, la partie restante servant déchantillons da validation. Pour evaluer la performance d'un algorithme, nous disposons de deux métriques,le premier étant le temps nécéssaire au processus d'apprentissage et de validation, et la second est la précision avec laquelle les résultats sont fiables. Une série de 50 tests ont été effectués sur les fichiers test data2.csv, data3.csv et data12.csv pour chaque learning ratio suivant : 0.2 , 0.4, 0.6 et 0.8. Les résultats ci-dessous présentes les moyennes sur ces 50 tests.

Pour reproduire les resultats observés il suffit de faire varier les paramètres dans le programme (cad le learning rate, nombre de test, affichage etc...). Par defaut la fonction testmain est configuré pour tester 50 fois l'opération suivante : Calcul de la durée et la précision de l'algorithme KNN et Bayes pour un learning ratio fixé. En redirigeant la sortie standard vers un fichier ( ex : python3 main.py > resultat.csv) il suffit ensuite de copier/coller correctement les résultats dans le tableur resultat_test.ods pour les résultat apparaissent. Les FIGURE 1, FIGURE 2, FIGURE 3 et FIGURE 4 ont été réalisé avec le logiciel GNUPlot avec les commandes suivantes :

## 5.1 Matrice de confusion

Afin d'analyser et de pouvoir comparer les résultats obtenus par les deux méthodes étudiées ici, on calcule pour chaque échantillon la matrice de confusion, permettant de déduire un taux d'erreur. Cette matrice est construite à l'aide d'une classe présentes dans la bibiliothèque Scikit-learn et en appliquant les calculs nécessaire pour calculer le taux d'erreur.

## 5.2 Jeu de test

Les jeux de test fournit montrent des classe distinct les uns des autre pour les fichiers data2.csv (voir FIGURE 1) et pour le fichier data3.csv (voir FIGURE 2, FIGURE 3 , FIGURE 4)

## 5.3 Observations

La FIGURE 5 nous montre que plus la proportion des données liés à l'apprentissage est basse ( cad plus la proportion lié à la validation est haute) plus la temps d'execution global de l'algorithme est long (cad temps d'apprentissage + temps de validation). La restitution des données pour la validation demande plus de temps de calcul que pour l'apprentissage. Cette difference de durée est observé aussi avec la méthode KNN (VOIR FIGURE 6) mais en beacoups moins marqué. Le pic correspondant à un learning ratio de 0.8 sur le jeu de test data12.csv de la FIGURE 6 reste inexpliqué.

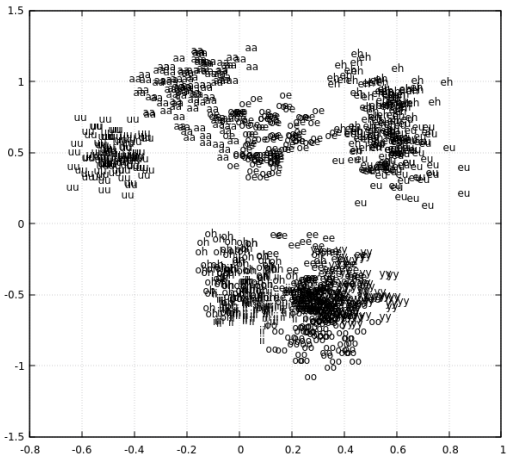

FIGURE 1 – Jeu de test data2.csv

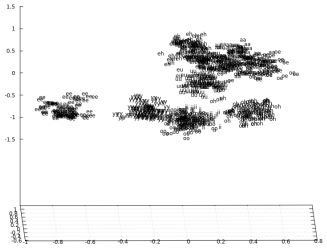

FIGURE 2 – Jeu de test data3.csv vue 1

Les FIGURE 7 et FIGURE 8 montre que sur ces jeux de test les méthode KNN et Bayes sont équivalentes, en effet la différence de précision entre ces deux méthode ne sont pas significative. Des données chiffrés sont disponible dans les FIGURE 9, FIGURE 10, FIGURE 11 et FIGURE 12.

**Learning ratio = 0.2.**

20 % des données ont été utilisé pour l'apprentissage de facons aléatoire pour chaque jeu de test (fichier.csv). Les

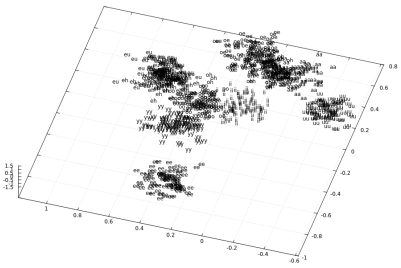

FIGURE 3 – Jeu de test data3.csv vue 2

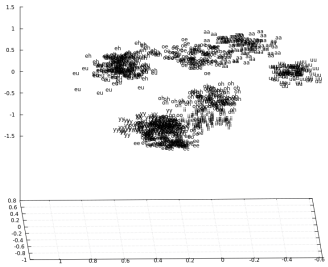

FIGURE 4 – Jeu de test data3.csv vue 3

données pour chaque jeu de test est une moyenne de 50 tests réalisés dans le mêmes conditions. Dans ce context nous observons que l'algorithme KNN est environ 40 fois plus rapide de celui de Bayes. La précision reste stable, KNN est très légerement moins précis que Bayes pour les deux premiers jeux de test (voir FIGURE 9).

**Learning ratio = 0.4.**

40 % des données ont été utilisé pour l'apprentissage de facons aléatoire pour chaque jeu de test (fichier.csv). Les données pour chaque jeu de test est une moyenne de 50 tests réalisés dans le mêmes conditions. Dans ce context nous observons que l'algorithme KNN est environ 60 fois plus rapide de celui de Bayes. La précision reste stable, KNN est très légerement plus précis que Bayes pour les deux derniers jeux de test (voir FIGURE 10).

**Learning ratio = 0.6.**

60 % des données ont été utilisé pour l'apprentissage de facons aléatoire pour chaque jeu de test (fichier.csv). Les données pour chaque jeu de test est une moyenne de 50 tests réalisés dans le mêmes conditions. Dans ce context nous observons que l'algorithme KNN est environ 80 fois plus rapide de celui de Bayes. La précision reste stable, KNN est très légerement plus précis que Bayes pour les deux derniers jeux de test (voir FIGURE 11).

**Learning ratio = 0.8.**

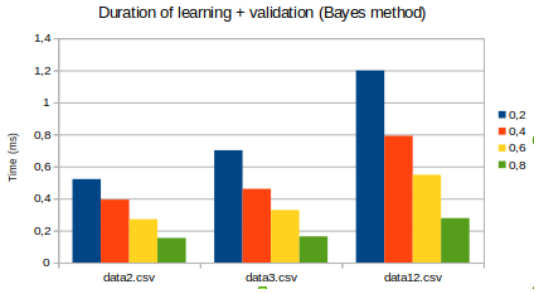


FIGURE 5 – Mesure du temps de l'algorithme Bayes sur les différents jeux de test pour chaque learning rate observé

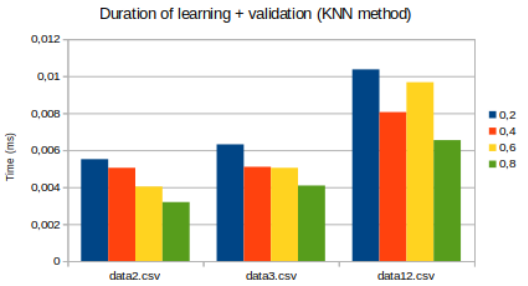


FIGURE 6 – Mesure du temps de l'algorithme KNN sur les différents jeux de test pour chaque learning rate observé

80 % des données ont été utilisé pour l'apprentissage de facons aléatoire pour chaque jeu de test (fichier.csv). Les données pour chaque jeu de test est une moyenne de 50 tests réalisés dans le mêmes conditions. Dans ce context nous observons que l'algorithme KNN est environ 110 fois plus rapide de celui de Bayes. La précision reste stable, KNN est très légerement plus précis que Bayes pour les deux derniers jeux de test (voir FIGURE 12).

# 6 Conclusion

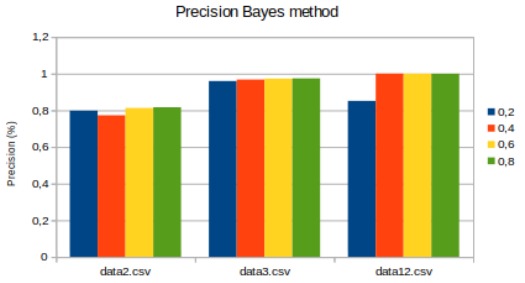


FIGURE 7 – Mesure du la précision de l'algorithme Bayes sur les différents jeux de test pour chaque learning rate observé

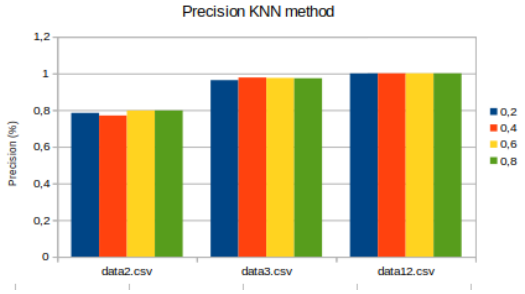


FIGURE 8 – Mesure du la précision de l'algorithme KNN sur les différents jeux de test pour chaque learning rate observé

| | Bayes | | KNN | | Evolution from Bayes to KNN | |
|---|---|---|---|---|---|---|
| Files test | Times | Precision | Times | Precision | Time Acceleration | Delta Precision |
| data2.csv | 0,153857116699 2 | 0,8165 | 0,00319028377 53 | 0,7976 | 48,226780918047 | -0,0189 |
| data3.csv | 0,163380889892 6 | 0,9739 | 0,00408449649 81 | 0,9733 | 40,000252165294 | -0,0006 |
| data12.csv | 0,277616457939 1 | 0,9997 | 0,00653800487 52 | 1 | 42,461953338842 | 0,0003 |

FIGURE 9 – Resultat pour 20% des données dédiées à l'apprentissage

| | Bayes | | KNN | | Evolution from Bayes to KNN | |
|---|---|---|---|---|---|---|
| Files test | Times | Precision | Times | Precision | Time Acceleration | Delta Precision |
| data2.csv | 0,270641374588 | 0,81255 | 0,00402904033 66 | 0,7971 | 67,172664450365 | -0,01545 |
| data3.csv | 0,328062300682 1 | 0,9727 | 0,00503987312 32 | 0,9749 | 65,093364984512 | 0,0022 |
| data12.csv | 0,547613954544 1 | 0,9995 | 0,00966653347 02 | 1 | 56,650500019978 | 0,0005 |

FIGURE 10 – Resultat pour 40% des données dédiées à l'apprentissage

| | Bayes | | KNN | | Evolution from Bayes to KNN | |
|---|---|---|---|---|---|---|
| Files test | Times | Precision | Times | Precision | Time Acceleration | Delta Precision |
| data2.csv | 0,392312622207 03 | 0,8080333333333 | 0,00504310131 07 | 0,7954333333333 | 77,791937519797 | -0,0126 |
| data3.csv | 0,460198983154 | 0,9706666666667 | 0,00585608959 2 | 0,973 | 76,584675163727 | 0,0023333333333 |
| data12.csv | 0,790948343277 | 0,9994 | 0,01086284637 45 | 1 | 72,812255278951 | 0,0006 |

FIGURE 11 – Resultat pour 60% des données dédiées à l'apprentissage

| | Bayes | | KNN | | Evolution from Bayes to KNN | |
|---|---|---|---|---|---|---|
| Files test | Times | Precision | Times | Precision | Time Acceleration | Delta Precision |
| data2.csv | 0,520898756980 9 | 0,79775 | 0,00550631046 3 | 0,7845 | 94,600324570451 | -0,01325 |
| data3.csv | 0,701043810844 4 | 0,9596 | 0,00630819797 52 | 0,96305 | 111,13218285239 1 | 0,00345 |
| data12.csv | 1,198473110199 | 0,8511 | 0,01036212921 14 | 1 | 115,65896214433 | 0,1489 |

FIGURE 12 – Resultat pour 80% des données dédiées à l'apprentissage