

# Rapport Projet M2: Classification de morceaux de verre pour la police scientifique

BARNACIN Endrick  
BASILE Johannie  
BRUTE Karl  
EUGÉNIE Reynald  
REMY Jamasa  
VOLET Djinny

<b>Introduction</b>	<b>3</b>
<b>Classification à l'aide des K Plus Proche Voisins</b>	<b>5</b>
Classification	5
Résultats	6
Précision du classifieur	6
Précision par classe	8
DISTANCE EUCLIDEAN	8
Distance de Manhattan	10
L'influence de K	12
Meilleur K	13
Conclusion	13
Autres études	14
Retour sur expérience	14
<b>Classification à l'aide du MLP</b>	<b>15</b>
Présentation du MLP	15
Expériences	15
Analyse des capacités de classification du réseau	18
Recherche du meilleur résultat	21
Conclusion	22
<b>Classification à l'aide d'autres classifieurs</b>	<b>23</b>
Classification avec les fenêtres de Parzen	23
Classification avec support vector machine	25
Classification avec des Arbres	29
<b>Outils de collaboration</b>	<b>35</b>
<b>Conclusion</b>	<b>35</b>
<b>Bibliographie / Webographie</b>	<b>36</b>
<b>Annexe</b>	<b>36</b>

# I. Introduction

Le problème de la classification consiste à retrouver la classe d'un exemple test grâce à une base d'apprentissage. Elle trouve de multiple applications dans tous les domaines de la vie courante tel que la médecine (détection de tumeurs), la biologie (reconnaissance des plantes et bien d'autres (reconnaissance des lettres, de la parole). Aussi de nos jours elle demeure une problématique importante pour les chercheurs en informatique et en mathématique. Ceux-ci testent leurs algorithmes une fois implémenté grâce à des jeux de données benchmark proposé par des tiers. Ces jeux de données benchmark permettent de connaître la véritable valeur d'un algorithme.

Plus particulièrement le jeu de données glass identification data set est un jeu de données réelles sur le verre.

Le verre est disponible sous différentes formes et différentes compositions chimique. On peut en trouver communément dans les fenêtres, les ustensiles de cuisine, ou encore les véhicules. La propriété du verre, particulièrement l'indice de réfraction, dépend de la composition et du traitement du verre. L'étude du problème de classification du verre a été motivée par une investigation criminologique.

Ce jeu contient donc plusieurs quantités d'éléments chimiques présents dans différents morceaux de verre dont nous savons le type. Il nous intéressera tout au long de ce rapport car nous tenterons de trouver le meilleur classifieur possible pour le détecter automatiquement.

Nous commencerons d'abord par présenter nos résultats avec l'algorithme du plus proche voisin, nous poursuivrons avec le perceptron multi-couche, enfin la dernière partie sera consacrée à d'autres classifieurs.

## II. Classification à l'aide des K Plus Proche Voisins

La **méthode des  $k$  plus proches voisins** est une méthode d'apprentissage supervisé.

Dans ce cadre, on dispose d'une base de données d'apprentissage constituée de  $N$  couples « entrée-sortie ». Pour estimer la sortie associée à une nouvelle entrée  $x$ , la méthode des  $k$  plus proches voisins consiste à prendre en compte les  $k$  échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée  $x$ , selon une distance à définir.

### Classification

Pour classifier les données, le classifieur compare la ressemblance et la dissemblance entre les différentes instances.

Deux instances sont semblables (appartiennent à la même classe) si la distance entre elles est faible ; réciproquement, deux instances sont dissemblables (appartiennent à des classes distinctes) si l'écart de distance entre eux est grande.

Pour classifier les bouts de verres, nous avons comparé les résultats obtenus grâce à deux types de distances, la distance de Manhattan et la distance Euclidienne.

#### DISTANCE DE MANHATTAN

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

#### DISTANCE EUCLIDEAN

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

## Résultats

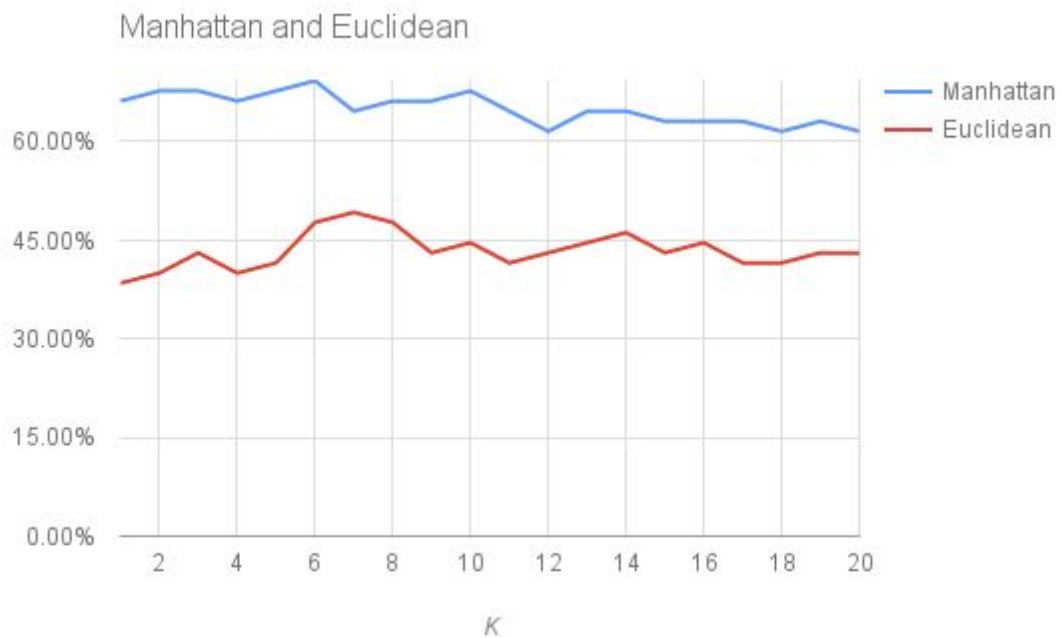
### Précision du classifieur

K	Manhattan	Euclidean
1	66.15%	38.46%
2	67.69%	40.00%
3	67.69%	43.08%
4	66.15%	40.00%
5	67.69%	41.54%
6	69.23%	47.69%
7	64.62%	49.23%
8	66.15%	47.69%
9	66.15%	43.08%
10	67.69%	44.62%
11	64.62%	41.54%
12	61.54%	43.08%
13	64.62%	44.62%
14	64.62%	46.15%
15	63.08%	43.08%
16	63.08%	44.62%
17	63.08%	41.54%
18	61.54%	41.54%
19	63.08%	43.08%
20	61.54%	43.08%

*Précision pour les deux variantes du classifieur pour K allant de 1 à 20*

	Manhattan	Euclidean
Moy	65.00%	43.38%
Min	61.54%	38.46%
Max	69.23%	49.23%

*Précision moyenne, minimum, maximum par classe.*



*Les précisions obtenu en utilisant la distance de Manhattan par rapport à la distance Euclidienne.*

Avec une précision minimale de 61.54 % et une précision moyenne de 65% contre une précision minimale de 38.46% et une précision moyenne de 43.38% pour le classifieur utilisant la distance Euclidienne, le classifieur utilisant la distance de Manhattan est la plus précise des deux versions des classifieurs que nous avons étudiés.

La classification avec la distance de Manhattan reste plus ou moins constante dans sa classification en restant au-dessus de la barre des 60 %.

A contrario avec l'augmentation de K la précision du classifieur utilisant la distance Euclidienne tant à osciller ce qui pourrait se transcrire comme une inconsistance dans la prise de décision.

## Précision par classe

Pour aller plus loin, nous avons décidé de regarder la précision du classifieur pour chaque classe. Dans le but de voir quelles sont les classes que le classifieur classe correctement et inversement quelles sont ceux, ils classent mal.

### DISTANCE EUCLIDEAN

K	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
1	57.89%	37.50%	0.00%	0.00%	0.00%	55.56%
2	84.21%	29.17%	0.00%	0.00%	0.00%	33.33%
3	68.42%	41.67%	0.00%	0.00%	0.00%	55.56%
4	78.95%	20.83%	0.00%	0.00%	0.00%	66.67%
5	78.95%	20.83%	0.00%	0.00%	33.33%	66.67%
6	84.21%	29.17%	20.00%	0.00%	33.33%	66.67%
7	84.21%	33.33%	20.00%	0.00%	33.33%	66.67%
8	84.21%	29.17%	20.00%	0.00%	33.33%	66.67%
9	73.68%	25.00%	20.00%	0.00%	33.33%	66.67%
10	84.21%	20.83%	20.00%	0.00%	33.33%	66.67%
11	73.68%	25.00%	20.00%	0.00%	0.00%	66.67%
12	84.21%	20.83%	20.00%	0.00%	0.00%	66.67%
13	84.21%	25.00%	20.00%	0.00%	0.00%	66.67%
14	84.21%	29.17%	20.00%	0.00%	0.00%	66.67%
15	84.21%	25.00%	0.00%	0.00%	0.00%	66.67%
16	84.21%	29.17%	0.00%	0.00%	0.00%	66.67%
17	84.21%	25.00%	0.00%	0.00%	0.00%	55.56%
18	78.95%	25.00%	0.00%	0.00%	0.00%	66.67%
19	84.21%	25.00%	0.00%	0.00%	0.00%	66.67%
20	84.21%	25.00%	0.00%	0.00%	0.00%	66.67%

*Précisions par classe.*

	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
<b>Moy</b>	80.26%	27.08%	9.00%	0.00%	10.00%	63.33%
<b>Min</b>	57.89%	20.83%	0.00%	0.00%	0.00%	33.33%
<b>Max</b>	84.21%	41.67%	20.00%	0.00%	33.33%	66.67%

*Précision moyenne, minimum, maximum par classe.*





### *Précision par classe : Euclidienne*

Le classifieur utilisant la distance Euclidienne parvient à classer les classes une et sept correctement avec une moyenne minimale de 63.33 % pour ces deux classes. Il échoue à classer correctement les classes deux, trois et six, mais ne parvient pas du tout à classer les instances de la classe cinq.

Les instances de la classe trois ont atteint une précision de classement maximale de 20 % pour  $K \in [6, 14]$  de même la classe atteinte un pic de 33.33 %  $K \in [5, 10]$  ce qui est très insatisfaisant.

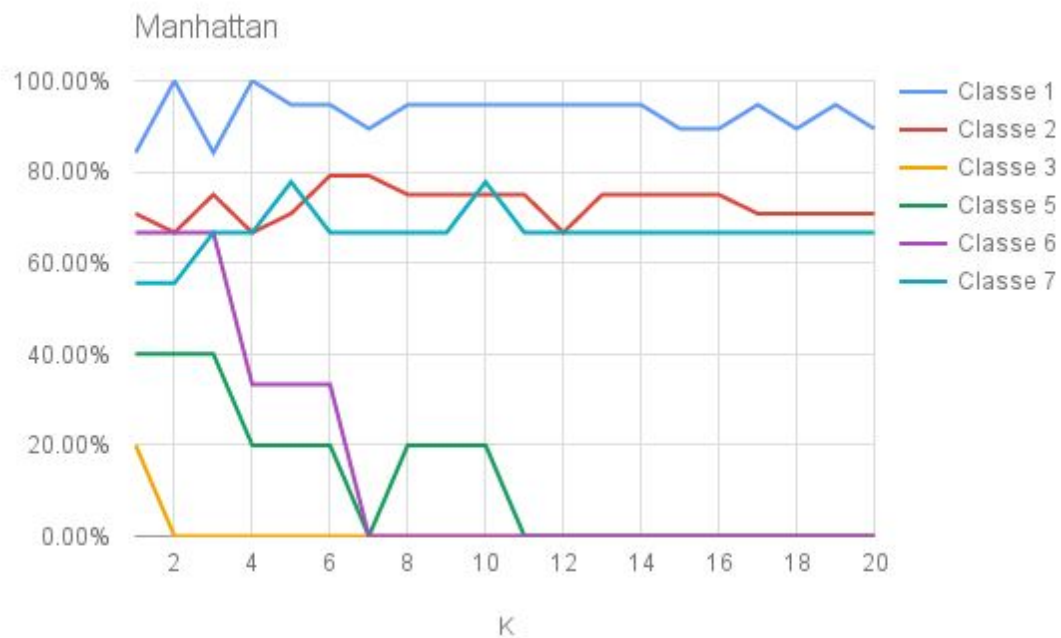
## Distance de Manhattan

K	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
1	84.21%	70.83%	20.00%	40.00%	66.67%	55.56%
2	100.00%	66.67%	0.00%	40.00%	66.67%	55.56%
3	84.21%	75.00%	0.00%	40.00%	66.67%	66.67%
4	100.00%	66.67%	0.00%	20.00%	33.33%	66.67%
5	94.74%	70.83%	0.00%	20.00%	33.33%	77.78%
6	94.74%	79.17%	0.00%	20.00%	33.33%	66.67%
7	89.47%	79.17%	0.00%	0.00%	0.00%	66.67%
8	94.74%	75.00%	0.00%	20.00%	0.00%	66.67%
9	94.74%	75.00%	0.00%	20.00%	0.00%	66.67%
10	94.74%	75.00%	0.00%	20.00%	0.00%	77.78%
11	94.74%	75.00%	0.00%	0.00%	0.00%	66.67%
12	94.74%	66.67%	0.00%	0.00%	0.00%	66.67%
13	94.74%	75.00%	0.00%	0.00%	0.00%	66.67%
14	94.74%	75.00%	0.00%	0.00%	0.00%	66.67%
15	89.47%	75.00%	0.00%	0.00%	0.00%	66.67%
16	89.47%	75.00%	0.00%	0.00%	0.00%	66.67%
17	94.74%	70.83%	0.00%	0.00%	0.00%	66.67%
18	89.47%	70.83%	0.00%	0.00%	0.00%	66.67%
19	94.74%	70.83%	0.00%	0.00%	0.00%	66.67%
20	89.47%	70.83%	0.00%	0.00%	0.00%	66.67%

*Précision par classe : Manhattan*

	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
<b>Moy</b>	92.89%	72.92%	1.00%	12.00%	15.00%	66.67%
<b>Min</b>	84.21%	66.67%	0.00%	0.00%	0.00%	55.56%
<b>Max</b>	100.00%	79.17%	20.00%	40.00%	66.67%	77.78%

*Précision moyenne, minimum, maximum par classe.*



*Précision par classe : Manhattan*

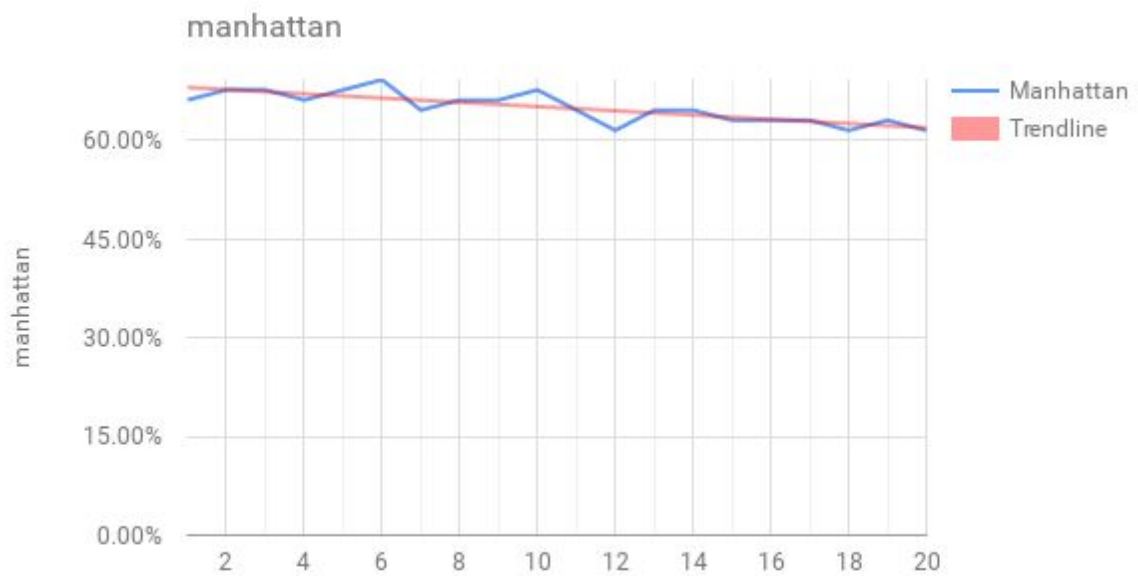
Par rapport à la classification basée sur la distance Euclidienne, on peut constater une nette amélioration des classements faits par le classifieur utilisant la distance de Manhattan.

Cette plus-value est visible dans le classement des instances de la classe une avec une précision maximale de 100 %, de la classe deux avec une précision de 79.92% pour la classe deux et une précision de 66.67 % pour la classe 7.

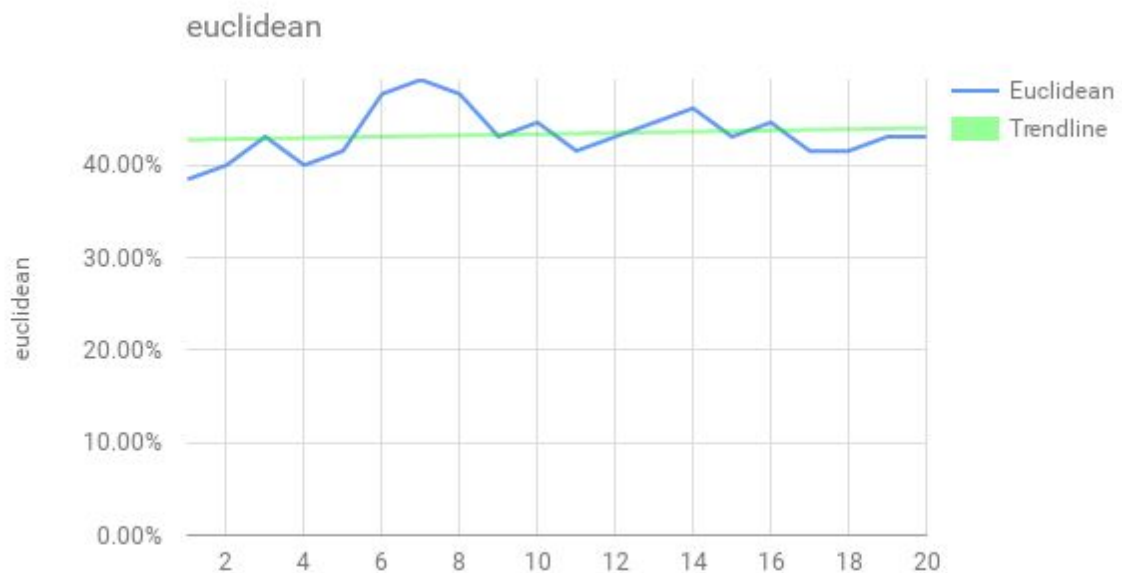
Ces résultats étant positifs et sont clairement une amélioration non-négligeable comparée aux résultats précédents, on se rend compte que même dans ce cas de figure le classifieur ne parvient pas à classer les instances issu de la classe trois, cinq et six.

Bien que ces classements restent là encore incontestablement meilleurs que les résultats obtenus en utilisant la distance Euclidienne. On ne peut que constater que le classifieur ne parvient pas à classer les instances de ces classes.

## L'influence de K



*Évolution de la précision vis-à-vis de K : Manhattan*



*Évolution de la précision vis-à-vis de K : Euclidienne*

En regardant l'évolution des deux courbes de précision au-dessus, nous pouvons constater que lorsque K augmente dans le cas de la distance de Manhattan la tendance de la précision (courbe rouge) est à la baisse alors que dans le cas de la distance Euclidienne (courbe verte) est à la hausse.

Néanmoins, cette tendance ne se reflète pas si l'on se réfère aux précisions obtenues pour chaque classe. Tel que mentionné précédemment les classes cinq, six et sept obtiennent une précision ne dépassant pas les 15 % en moyenne. Leurs instances sont donc très mal classées, elles se retrouvent dans des classes qui ne leur appartiennent. Cela contredit le phénomène observé concernant la tendance de la précision du KPPV utilisant la distance Euclidienne.

Ce qui est certain, c'est que l'augmentation du K dégrade la précision et cela est d'autant plus visible dans le cas du KPPV utilisant la distance de Manhattan.

## Meilleur K

Pour les deux types de distances utilisées en combinaison avec le classifieur KPPV que nous avons utilisé durant cette étude, nous avons retenu deux valeurs pour K pour lesquelles nous avons obtenu les meilleurs classements sont K = 1 pour la distance de Manhattan et K = 7 pour la distance Euclidienne.

K	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
1	84.21%	70.83%	20.00%	40.00%	66.67%	55.56%

*Meilleur K pour la distance de Manhattan*

K	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
7	84.21%	33.33%	20.00%	0.00%	33.33%	66.67%

*Meilleur K pour la distance Euclidienne*

## Conclusion

Des deux types de distances utilisées dans cette étude de classification à l'aide de l'algorithme des K Plus Proches Voisins, celui qui a donné les meilleurs résultats est sans nuls doutes la distance de Manhattan.

La classification des instances des classes n'est pas uniforme pour toutes les classes, les classes une, deux et trois sont ceux qui ont les meilleurs taux de classement avec respectivement 92.89 %, 72.92 % et 66.67 % en moyenne (KPPV + Manhattan).

L'influence de K tant à dégrader la précision du classifieur alors qu'elle grandit, cette dégradation de la précision du classifieur est perceptible au niveau des classes et ne l'ai pas énormément si l'on se fie à la précision global du classifieur.

En somme même en utilisant le meilleur K couplé avec la meilleure distance en l'occurrence la distance de Manhattan ; avec les résultats obtenus, la précision du classifieur n'est pas suffisamment fiable pour permettre de tirer des conclusions acceptables.

## Autres études

Mashael S. Aldayel, chercheur à l'université King Saud, a réalisé une étude sur ce même jeu de données dans le but de montrer qu'utiliser plusieurs classifieurs offre de meilleur résultats que l'utilisation d'un classifieur unique.

Il a dans un premier temps, comparer les résultats sur plusieurs méthode d'implémentation de l'algorithme du k plus proche voisin à partir de travaux déjà réalisés.

Avant d'entamer la classification, il a cherché à pré-traiter le jeu de données en nettoyant les données, réduisant les dimensions et en effectuant la normalisation, la transformation et la discrétisation. Ceci lui a permis de voir que les caractéristiques Iron et Silicon sont les moins significatif dans la composition du verre.

la deuxieme phase comprend l'utilisation de classificateurs simples et de classificateurs multiples (votant avec KPPV et HNB Hidden Naive Bayes ) pour construire un modèle de prédiction à haute précision pour le problème d'identification du verre

# III. Classification à l'aide du MLP

## Présentation du MLP

Le multilayer perceptron ( ou MLP ) est un réseau de neurones formel proposé par Paul Werbos en 1984 et mis en place par David Rumelhart en 1986.

Basé sur le perceptron de Frank Rosenblatt , le MLP est un réseau de neurones comportant plusieurs couches dont une couche d'entrée recevant les informations de l'exemple à classer, une ou plusieurs couches cachées par lesquels vont transiter l'information et une couche de sortie indiquant la classe dans laquelle l'information sera classifiée .

Les neurones de chaque couches sont connectés à tous les neurones de la couche précédente.

Le MLP est un réseau de type feedforward. Cela signifie que l'information ne circule que dans un sens , les neurones de chaque couche reçoivent l'information envoyée par les neurones de la couche précédente et calculent alors leurs valeurs de sortie. Les liaisons entre les différents neurones sont soumises à des coefficients qui déterminent l'effet de l'information sur le neurone. Les neurones transmettent leurs valeurs de sortie aux neurones de la couche suivante .

La phase d'apprentissage est géré par un algorithme de rétropropagation .

Cet algorithme consiste en 8 étapes qui sont les suivantes :

- 1 - Présentation d'un motif d'entraînement au réseau
- 2 - Comparaison de la sortie du réseau avec la sortie ciblée
- 3 - Calcul de l'erreur en sortie de chacun des neurones du réseau
- 4 - Calcul, pour chacun des neurones, de la valeur de sortie qui aurait été correcte
- 5 - Définition de l'augmentation ou de la diminution nécessaire pour obtenir cette valeur (erreur locale)
- 6 - Ajustement du poids de chaque connexion vers l'erreur locale la plus faible
- 7 - Attribution d'un blâme à tous les neurones précédents
- 8 - Recommencer à partir de l'étape 4, sur les neurones précédents en utilisant le blâme comme erreur

## Expériences

Pour pouvoir tester et évaluer les performances du MLP, nous avons utilisé le programme “ MLP-MULTIclasses\_prog ” écrit par Madame PAUGAM-MOISY.

L'utilisation d'un réseau MLP requiert de déterminer certains paramètres :

- Le nombre de passes de la base d'apprentissage
- La valeur du pas du gradient
- Le nombre de couches cachées
- Le nombre de neurones par couche cachée

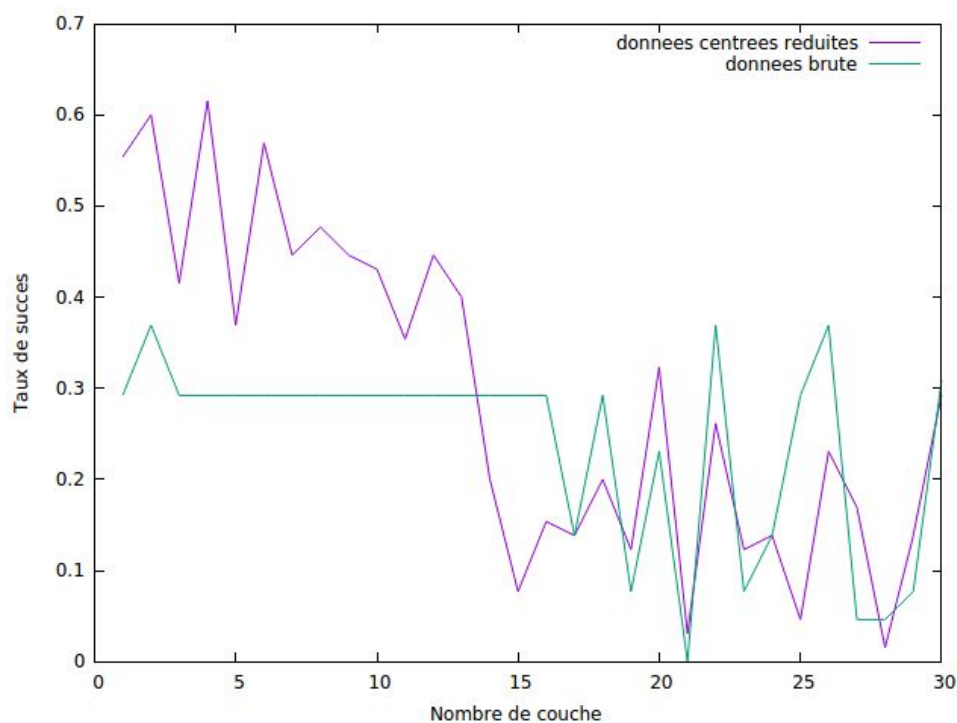
La valeur de ces paramètres détermine le taux de succès de cet algorithme sur les bases d'apprentissage (taux d'apprentissage) et de test (taux de généralisation).

Notre analyse consistera donc à déterminer quelles sont les valeurs pour lesquelles le réseau MLP obtient les meilleurs taux de classification.

Pour commencer, nous allons tester l'influence de la modification du nombre de couches cachées du réseau. Pour ce faire, nous avons fixé les autres paramètres et fait varier le nombre de couches cachées afin d'enregistrer les variations sur le taux de réussite en généralisation.

Les courbes ci-dessous montrent l'évolution du taux de généralisation du classifieur sur les données brutes mais aussi sur les données centrées et réduites.

Nous fixons le nombre de neurones par couche cachée à 15 et le nombre de passes sur la base d'apprentissage à 1000 .



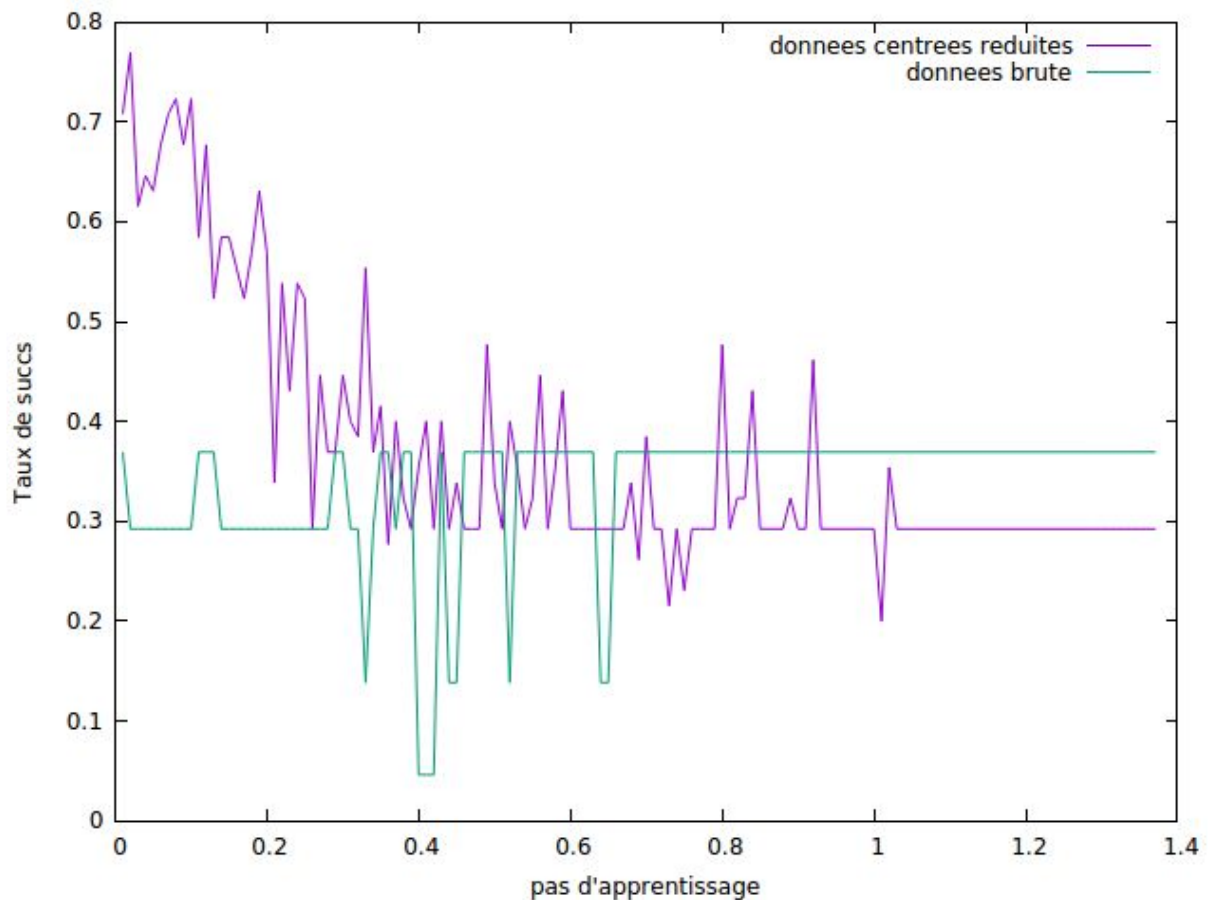
Les meilleurs résultats sont obtenus avec une seule couche cachée et le taux de généralisation se dégrade lorsque l'on augmente le nombre de couches.

À partir du test précédent, nous savons que pour de bonnes performances nous ne conserverons qu'une seule couche cachée.

Nous allons donc tenter de faire varier la valeur du pas du gradient.



Le nombre de neurones restera fixé à 15 et le nombre de passes de la base d'apprentissage à 100 .



Evolution du taux de succès vis-à-vis du pas d'apprentissage

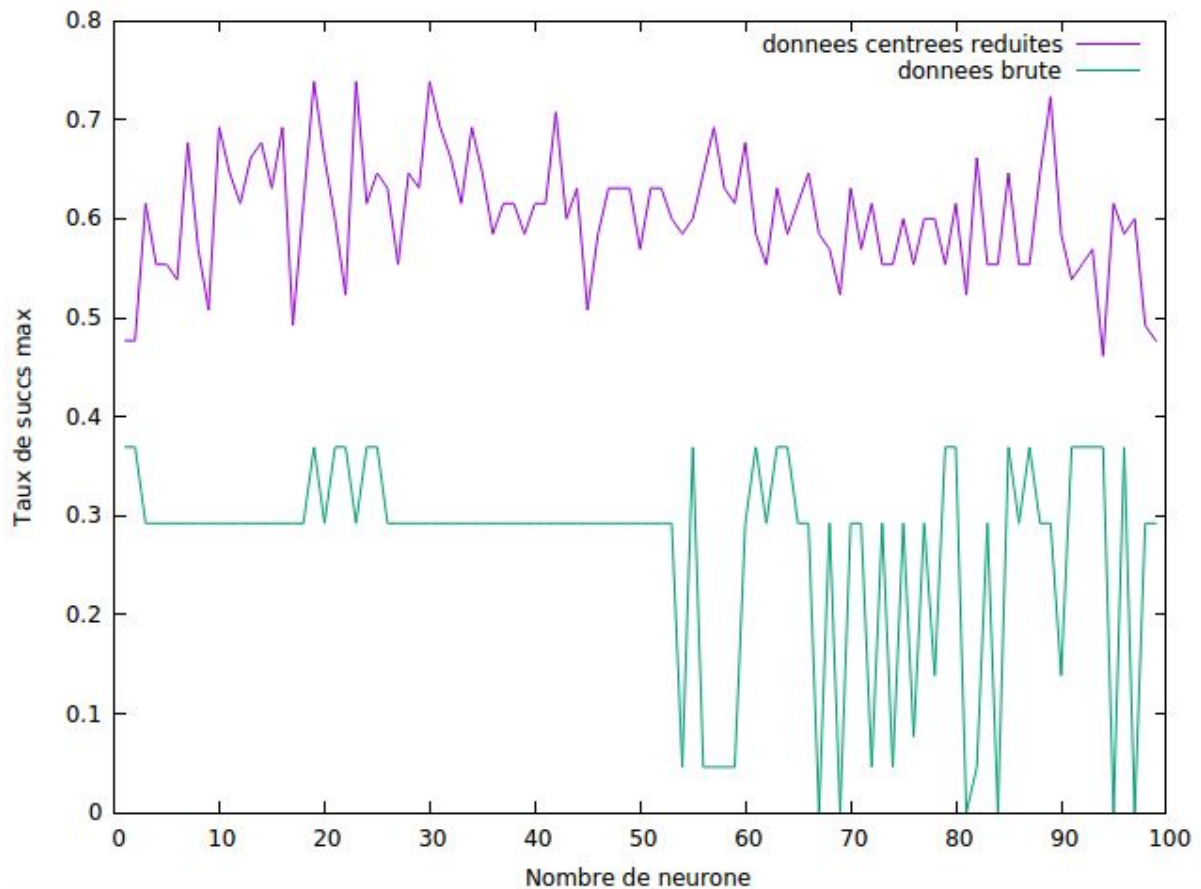
Sur la courbe ci-dessus nous voyons que les meilleurs résultats sont obtenus pour les données centrées réduites et un pas d'apprentissage inférieur à 0,1. Au-delà, les résultats faiblissent jusqu'à trouver un point plateau de 0,3 pour les données centrées réduites et de 0,4 pour les données brutes.

Nous avons ensuite observé les performances du MLP en variant le nombre de neurones sur la couche cachée.

Pour ce test, nous avons conservé un pas de gradient 0.05 car l'expérience 1 montre de meilleurs résultats pour un pas inférieur à 0.1.

Le nombre de couche cachée : 1

Le nombre de passes de la base d'apprentissage : 1000



Évolution du taux de succès par rapport au nombre de neurones

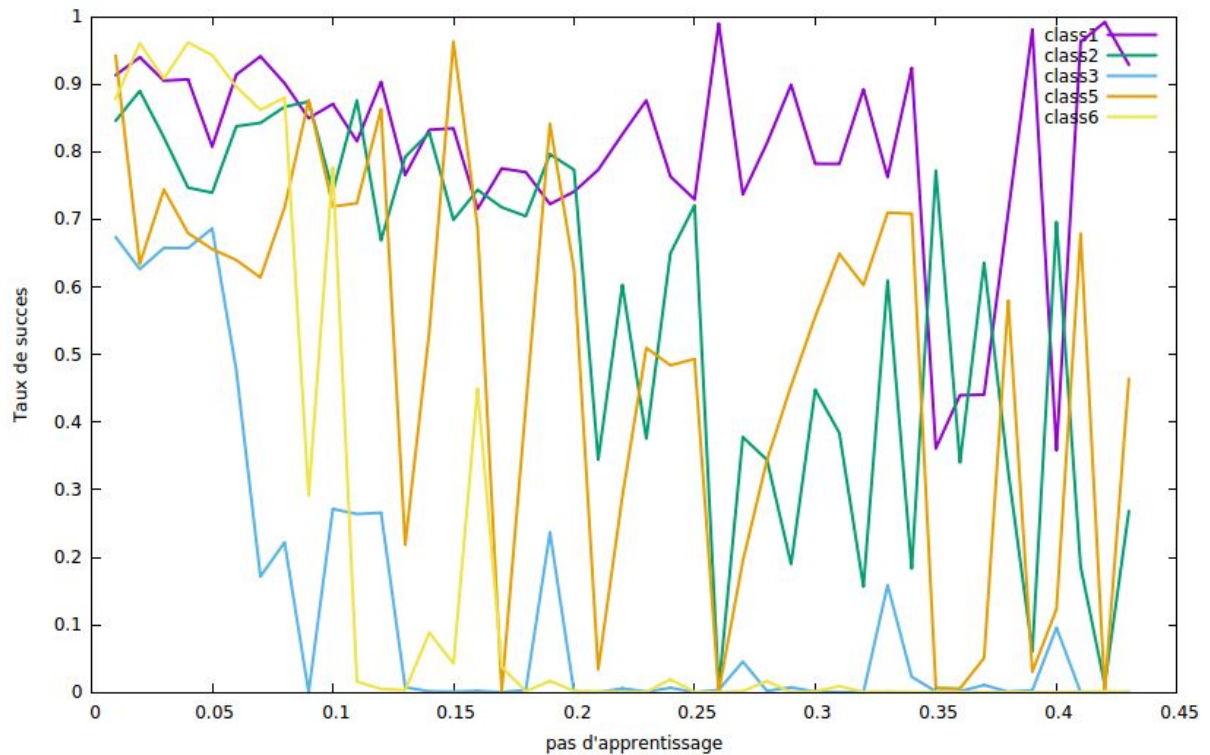
A la suite de cette expérience nous constatons une fois encore que les données centrées réduites sont beaucoup plus performantes que les données brutes. Le taux de succès quant à lui oscille entre 0,5 et 0,7.

## Analyse des capacités de classification du réseau

Les taux de généralisation obtenus avec le réseaux MLP sont environ compris entre 0.5 et 0.7 , il serait donc intéressant d'observer la capacité du réseau MLP à classier les exemples dans les classes appropriées .

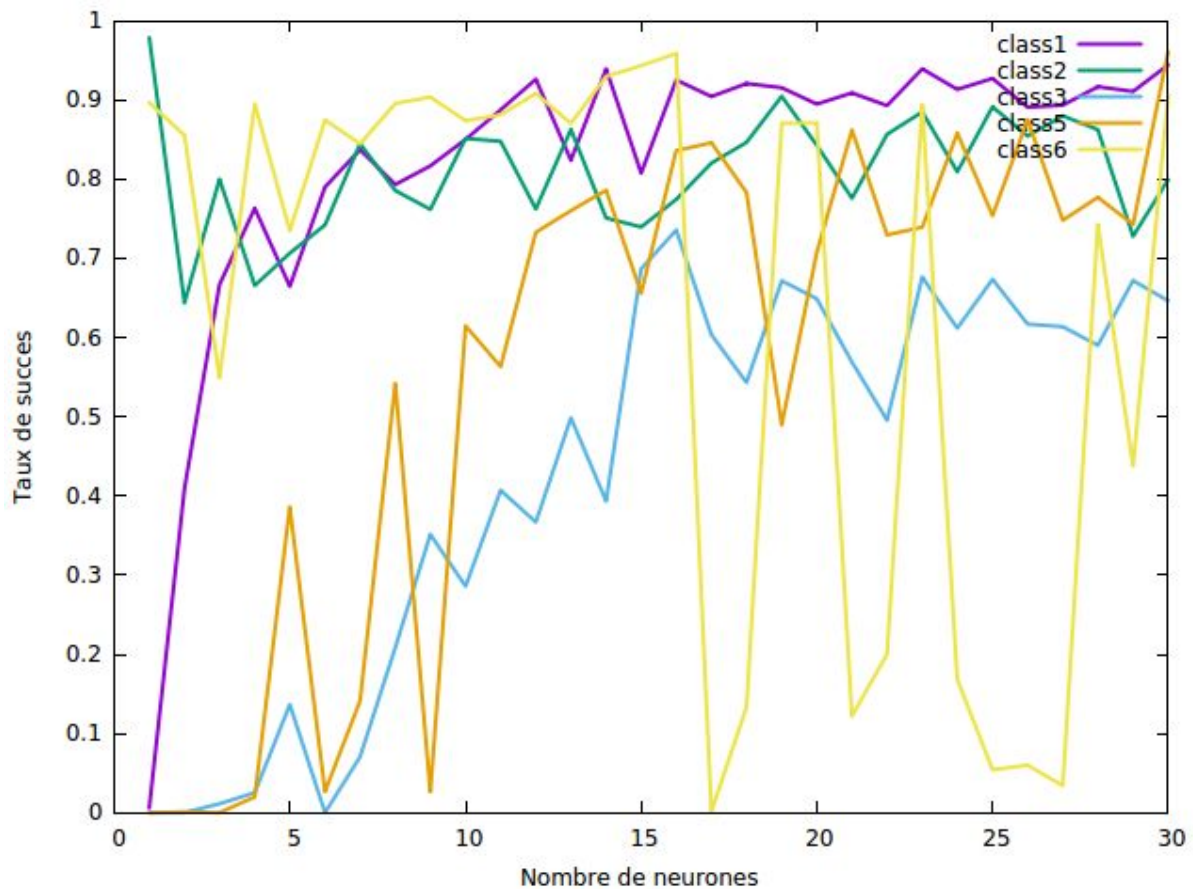
Pour ce faire nous avons récupéré les sorties du réseau MLP et vérifié si la classe déterminée par le réseau correspond à la classe d'origine de l'exemple.

Nous obtenons les graphes suivants:



Évolution du taux de succès par rapport au pas d'apprentissage pour les données centrées réduites (15 neurones, 1 couche)

Nous remarquons ci-dessus que l'augmentation du pas d'apprentissage affecte négativement la capacité du réseau à classer correctement les classes. Les classes 3 et 6 sont celles qui sont les moins bien classées par le réseau lorsque le pas augmente tandis que la classe 1 est celle qui produit le meilleur score.



Évolution du taux de succès par rapport au nombre de neurones pour les données centrées réduites (0.05 pas d'apprentissage, 1 couche)

La répartition du taux de succès selon la classe est très chaotique pour des nombres de neurones différents. Ce schéma ne nous permet pas de tirer grande conclusion si ce n'est que pour les classes 1 et 2 la précision du réseau de neurone semble être meilleure. Ces classes seraient donc plus faciles à classifier que les autres. Enfin nous concluons en disant tout simplement que le nombre de neurone affecte énormément les frontières de séparation des classes que le MLP construit au cours de la rétropropagation du gradient.

## Recherche du meilleur résultat

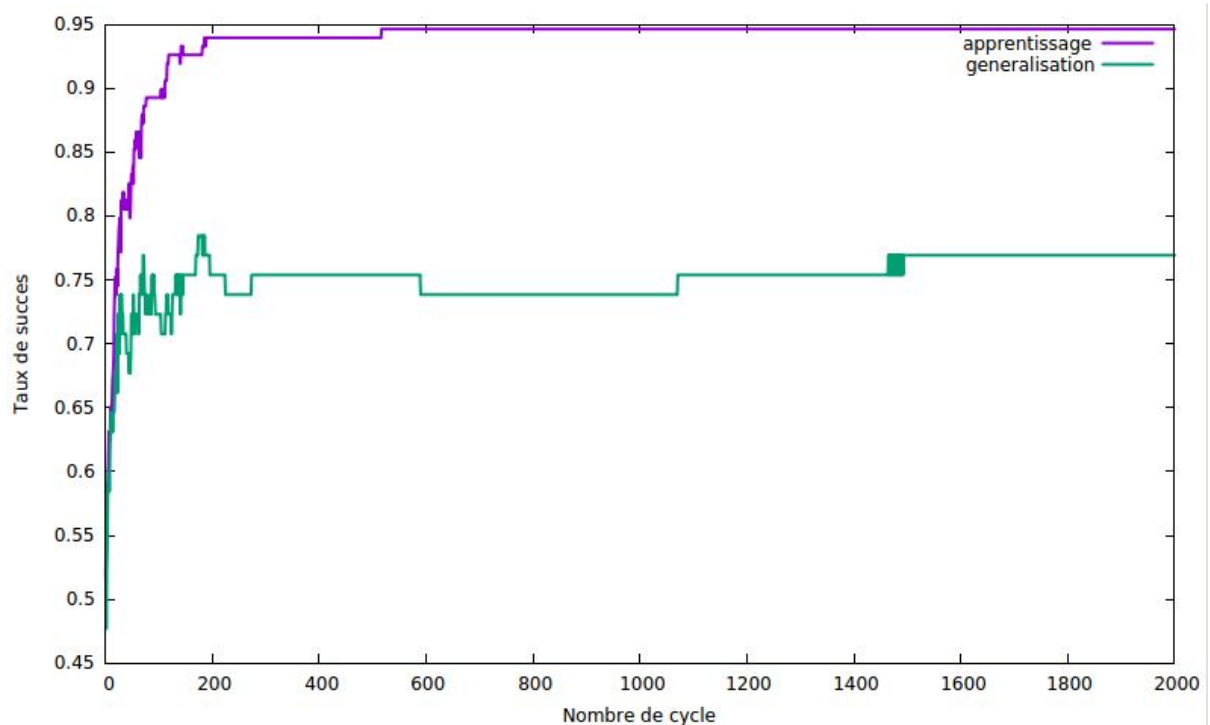
À partir de toutes les données collectées, nous avons tenté de déterminer les paramètres pour lesquels le taux de généralisation est le meilleur.

Nous savons que le pas du gradient doit être inférieur à 0.1, qu'il ne doit y avoir qu'une seule couche cachée et que le nombre de neurones par couche doit être faible.

Nous sommes donc arrivés au résultat suivant :

Valeur du pas du gradient : 0.02

Nombre de neurones par couche cachée : 15



Avec ces paramètres le taux de généralisation atteint 0.769231 (soit approximativement 77%) pour un taux d'apprentissage atteignant 0.94 .

Les taux d'apprentissage et de généralisation se stabilisent après seulement 300 passes et le taux de généralisation atteint sa valeur maximale après 1400 passes .

## Conclusion

En conclusion les données qui obtiennent le meilleur taux de réussite avec le perceptron multi-couche sont les données centrées réduites. Les classes pour lesquelles nous obtenons le meilleur taux de réussite sont les classe 1 et 2.

Les différents paramètres du réseau MLP influent grandement sur ces capacités en généralisation. Il est donc important de déterminer les valeurs pour lesquels le réseau obtient le meilleur taux de classification. Or, ces valeurs diffèrent selon les exemples contenus dans la base d'apprentissage.

Le réseau MLP offre des résultats stables mais dépendant des paramètres sélectionnés pour un apprentissage efficace .

## IV. Classification à l'aide d'autres classifieurs

### Classification avec les fenêtres de Parzen

#### Présentation de l'algorithme des fenêtre de Parzen:

En statistique, l'estimation par noyau (ou encore méthode de Parzen-Rosenblatt) est une méthode non-paramétrique d'estimation de la densité de probabilité d'une variable aléatoire. Elle se base sur un échantillon d'une population statistique et permet d'estimer la densité en tout point du support. En ce sens, cette méthode généralise astucieusement la méthode d'estimation par un histogramme.

Dans le classifieur à fenêtre de parzen, nous estimons la densité de probabilité qu'un exemple appartient à une classe et classifions cet exemple dans la classe pour laquelle il obtient la plus haute densité de probabilité.

L'estimation de la probabilité d'appartenance à une classe se fait grâce à la formule suivante:

$$P(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{h_n^d} K\left(\frac{x - x_i}{h_n}\right)$$

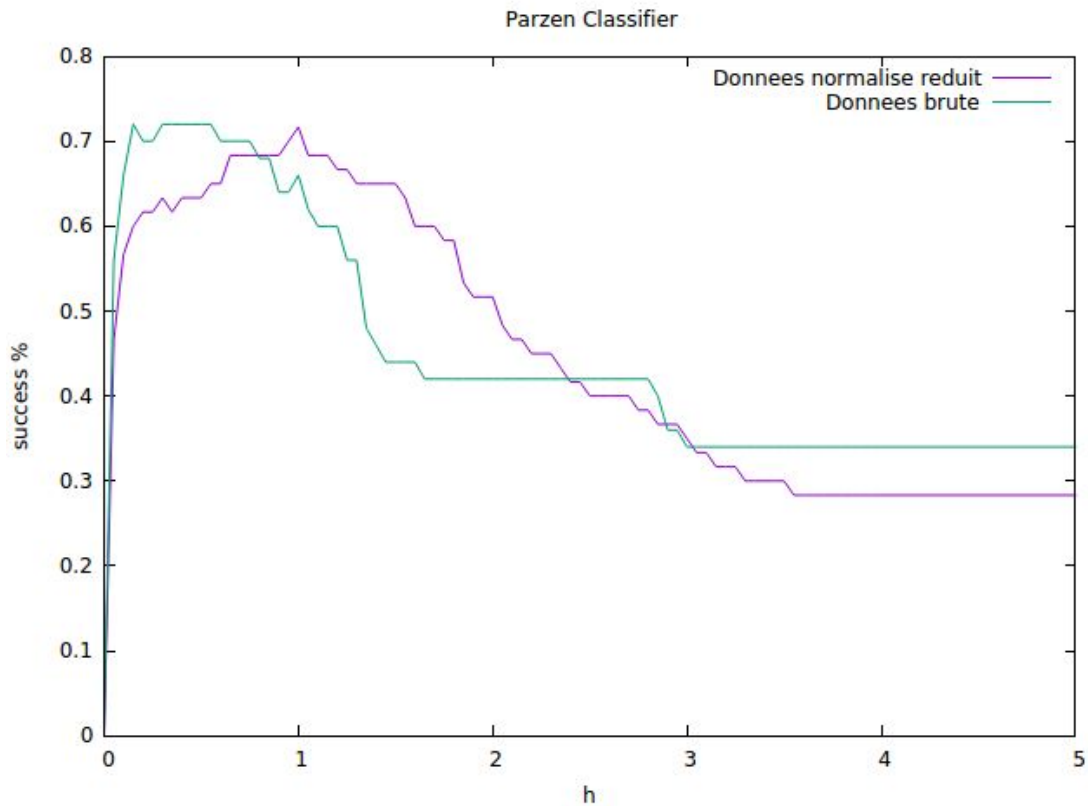
ou  $n$  est le nombre d'exemple  $h$  est le paramètre de lissage et ou  $K$  est la fonction fenêtre aussi appelé noyau. En utilisant un noyau gaussien nous estimons la probabilité d'appartenance à une classe via la formule suivante:

$$P(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{(h\sqrt{2\pi})^d} \exp\left(-\frac{1}{2}\left(\frac{x - x_i}{h}\right)^2\right)$$

Si le choix du noyau est réputé comme peu influent sur l'estimateur, il n'en est pas de même pour le paramètre de lissage. Un paramètre trop faible provoque la prise en compte de détails artificiels lors de la classification. Pour une valeur de  $h$  trop grande, la majorité des caractéristiques est au contraire effacée. Le choix de  $h$  est donc une question centrale dans l'estimation de la densité.

#### Influence du paramètre de lissage $h$ sur le taux de succès

Noyau:gaussien



Évolution du taux de succès par rapport au paramètre de lissage h

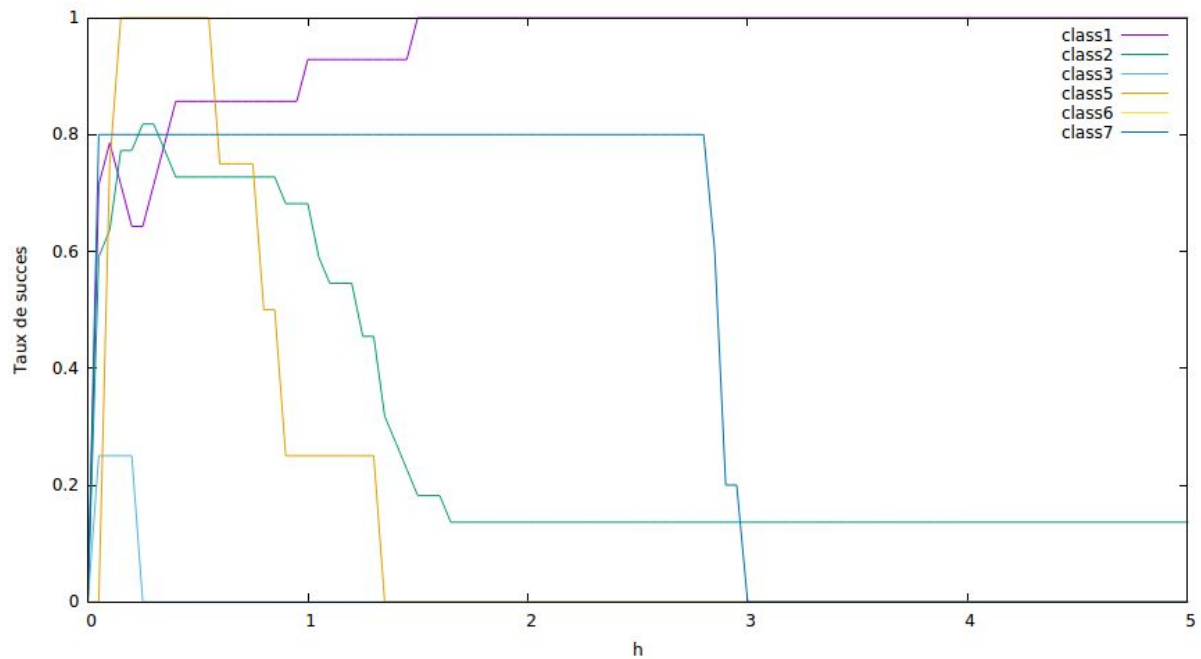
La figure ci-dessus nous montre le taux de succès en utilisant le classifieur parzen et en faisant varier le paramètre de lissage h. Nous remarquons que la performance maximale obtenu est de 72 % pour les données brute et pour les données centrées réduites. Notons également qu'au delà d'un certain h il n'y a plus de variation des résultat et que le h pour lequel nous obtenons le maximum de performance pour les données centrées réduites est 1 tandis que celui pour lequel nous atteignons le maximum de performance pour les données brutes est 0,15.

	Classe 1	Classe 2	Classe 3	Classe 5	Classe 6	Classe 7
h=0.15	71,5%	77.27%	25.00%	1.00%	0.00%	80.00%

Évolution du taux de succès pour chaque classe par rapport au paramètre de lissage h pour les données centrées réduites.

Nous observons la répartition du taux de succès ci-dessus pour chaque classe ci-dessus pour le h optimum trouvées et les données brutes. Les meilleures classifications sont obtenus pour la classe 1, 2 et 7.





Sur la courbe ci-dessus nous pouvons constater que la classification des exemples de chaque classe diffère selon le  $h$ . Notons que la taux de succès obtenus pour la classe 1, est le seul qui progresse quand  $h$  croît au point d'atteindre 100% de succès.

Pour conclure bien que ce classifieur présente un taux de succès supérieur à celui du plus proche voisin, celui-ci à un taux de succès inférieur à celui du réseau de neurones. Le perceptron multicouche semble être bien plus adapté à notre problème.

## Classification avec support vector machine

### Présentation des Support Vector Machine:

La support vector machine est capable de traiter des problèmes de discrimination non linéaire. Elle transforme l'espace de représentation de nos données en un espace de plus grande dimension dans lequel une séparation linéaire de nos données existe. Cela fait, elle cherchera alors à maximiser la frontière de séparation entre les données les plus proches et de classes différentes.

Le problème est de trouver cette frontière séparatrice optimale, à partir d'un ensemble d'apprentissage. Ceci est fait en formulant le problème comme un problème d'optimisation quadratique. Problème pour lequel il existe de nombreux algorithmes de résolution.

### Principe générale:

Le cas simple est le cas d'une fonction discriminante linéaire, obtenue par combinaison linéaire du vecteur d'entrée  $x = (x_1, \dots, x_N)^T$ , avec un vecteur de poids  $w = (w_1, \dots, w_N)^T$  :

$$h(x) = w^T x + w_0$$

Il est alors décidé que  $x$  est de classe 1 si  $h(x) \geq 0$  et de classe -1 sinon. C'est un classifieur linéaire.

La frontière de décision  $h(x) = 0$  est un hyperplan, appelé *hyperplan séparateur*, ou *séparatrice*. Le but d'un algorithme d'apprentissage supervisé est d'apprendre la fonction  $h(x)$  par le biais d'un ensemble d'apprentissage :

$$\{(x_1, l_1), (x_2, l_2), \dots, (x_k, l_k), \dots, (x_p, l_p)\} \subset \mathbb{R}^N \times \{-1, 1\}$$

où les  $l_k$  sont les labels,  $p$  est la taille de l'ensemble d'apprentissage,  $N$  la dimension des vecteurs d'entrée. Si le problème est linéairement séparable, on doit alors avoir :

$$l_k h(x_k) \geq 0 \quad 1 \leq k \leq p, \quad \text{autrement dit} \quad l_k (w^T x_k + w_0) \geq 0 \quad 1 \leq k \leq p.$$

(Source:Wikipédia)

Pour faire nos expérience sur les support vector machine nous avons choisi d'utiliser la libSVM, par soucis de simplicité, de fiabilité et car celle-ci est open Source . Nous utiliserons la méthode de la k-fold cross validation implémentée dans cette librairie afin d'obtenir nos résultats.

### Expérience 1 Comparaison des noyaux:

cost =5

#### Résultat:

Nom noyaux	Formule	Taux de succès
linear	$u \cdot v$	64.5%
polynomial	$(\gamma u \cdot v + \text{coef0})^{\text{degree}}$	53%
radial basis function	$(-\gamma  u-v ^2)$	69%
sigmoid	$\tanh(\gamma u \cdot v + \text{coef0})$	43.5%

FIG: Taux de succès selon le noyau choisi

Le noyau avec lequel l'on obtient le meilleur taux de succès est le noyau gaussien. Ce résultat est peu surprenant, car le noyau gaussien est le noyau qui correspond le plus souvent aux jeux de données réelles. Remarquons tout de même que nous obtenons un bon taux de réussite avec le noyau linéaire. Nous avons donc choisi de continuer nos expérimentations avec le noyau gaussien.

### Expérience 2 : Influence de l'hyper-paramètre gamma avec cost =5

L'hyper-paramètre gamma permet au noyau gaussien de gérer des classifications non linéaire.

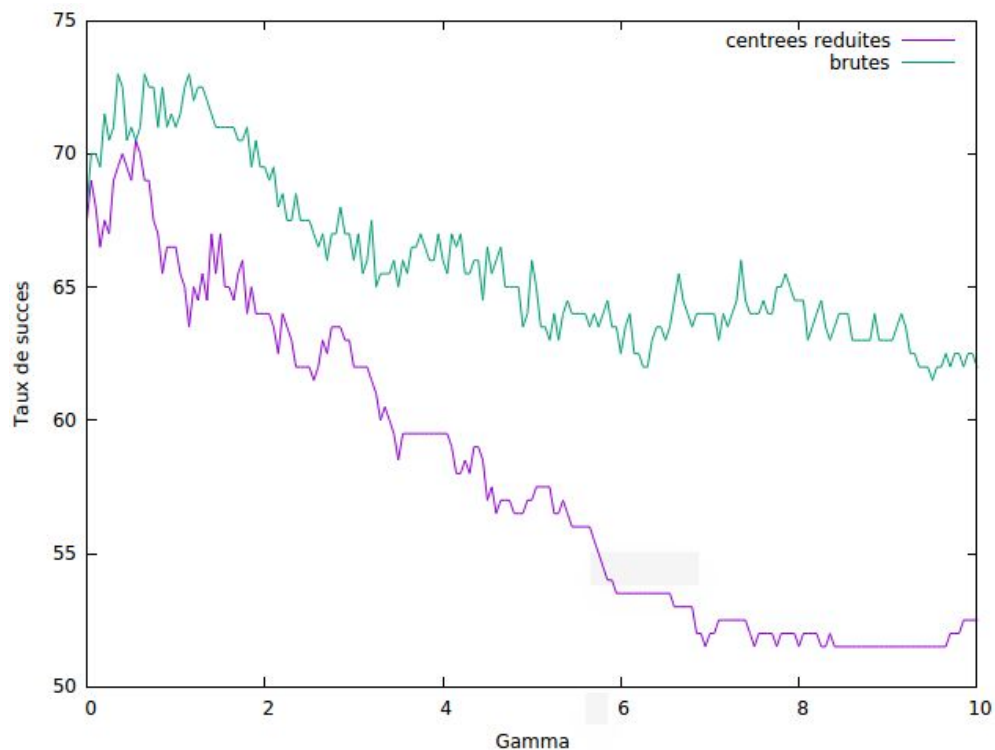


FIG : Évolution du taux de succès par rapport à l'hyper-paramètre gamma

Sur la figure ci-dessus nous remarquons que les données centrées réduites sous performent les données brutes. Nous remarquons également que les meilleurs résultats sont obtenus pour une petit ( $<2$ ) gamma . Au-delà nos résultats sont insatisfaisants.

### Expérience 3 : Influence de l'hyper-paramètre cost avec gamma=0.5

Le paramètre cost est le paramètre qui permet d'indiquer jusqu'à quel point nous pouvons accepter une mauvaise classification. Pour une grande valeur, la SVM choisira une faible distance de marge entre les hyperplans qui séparent nos classes. A contrario une valeur cost petite augmentera la marge entre nous hyperplan et poussera le classifieur à se tromper plus souvent sur les exemples de la base d'entraînement.

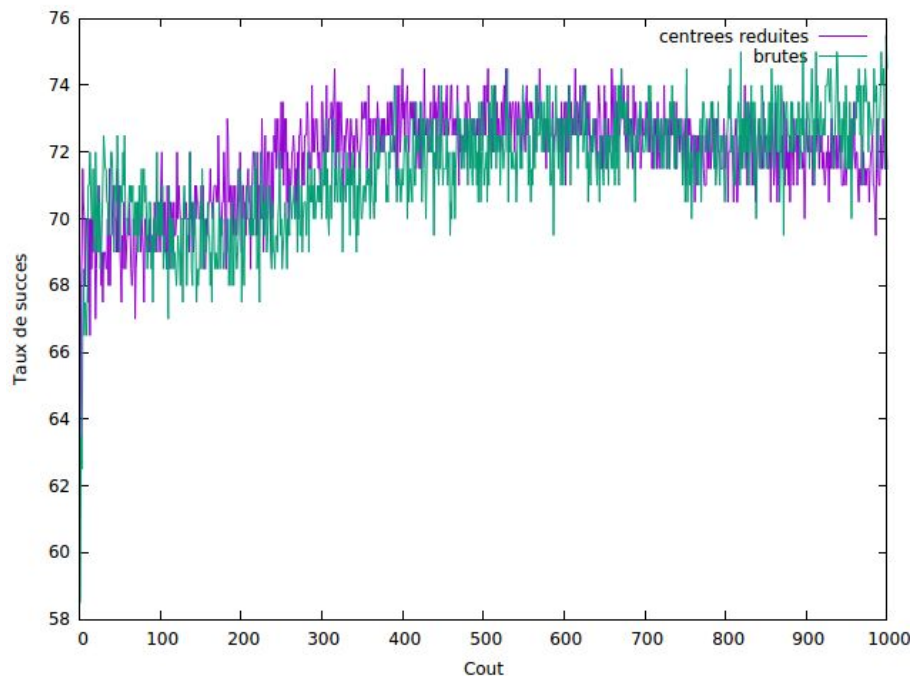


FIG : Évolution du taux de succès par rapport à l'hyper-paramètre gamma

Sur le graphique ci-dessus nous voyons qu'entre 150 et 700 nous obtenons de meilleurs résultats avec les données centrées réduites. Mais c'est avec les données brutes que nous obtenons les meilleurs résultats finaux. Notons que le taux de succès maximal obtenu est 75,5 % pour un paramètre cost de 999.

Pour conclure nous dirons que la SVM ne présente certes pas un taux de réussite de 100% pour notre jeu de données mais est plutôt robuste (75.5% en k-fold cross validation ) pour notre tâche de classification.

## Classification avec des Arbres

L'arbre de décision est un outil qui a l'avantage d'être facile à exécuter. Dans cet arbre, les noeuds contiennent des conditions permettant d'atteindre les feuilles à partir d'un objet afin de déterminer sa classe. Il existe plusieurs moyens de créer un arbre de décision, aussi nous avons fait des tests sur plusieurs d'entre eux en utilisant l'outil de classification Weka.

### J48 Tree

Le méthode J48 utilise l'algorithme C4.5 afin de générer un arbre en utilisant l'entropie. Sur les différents paramètres proposés par l'outil, nous avons estimé que le plus pertinent est minNumObj, le nombre minimum d'objet qu'une feuille doit avoir. Néanmoins la classe 6 étant représenté par 3 objets uniquement, nous avons décidé de ne pas élever ce paramètre au delà de 4. Nous obtenons les résultats suivant pour les différents minNumObj :

minNumObj	Clas.1	Clas.2	Clas.3	Clas.5	Clas.6	Clas.7	Succès
2	57,89	79,16	60	80	66,66	66,66	69,23
3	73,68	79,16	20	80	66,66	77,77	72,3
4	68,42	87,5	20	60	66,66	77,77	72,3

Nous constatons que le changement est assez important si le minNumObj est à 2 plutôt que 3 ou 4, mais que la différence entre les classifieurs utilisant 3 ou 4 valeurs minimum par feuille est moindre. Il est difficile d'établir quel classifieur est le plus intéressant parmi ceux que nous avons obtenu car d'un côté nous avons des classifieurs avec un taux de vrai positif de seulement 20% pour la classe 3 mais des taux de succès généraux atteignant 72.3% face à un classifieur dont le taux moyen est de 69.23% mais qui a un taux de vrai positif de plus de 50% pour chaque classe.

### J48 Graft Tree

La méthode J48 Graft, comme on peut le deviner, est assez proche de la méthode J48 sauf que cette dernière va greffer dans l'arbre J48 des noeuds dans le but d'affiner les résultats

obtenus. Après l'utilisation de cet algorithme en faisant varier une fois encore le minNumObj, nous avons pu obtenir le tableau de résultats suivant :

minNumObj	Clas.1	Clas.2	Clas.3	Clas.5	Clas.6	Clas.7	Succès
2	57,89	83,33	40	80	66,66	77,77	70,76
3	73,68	83,33	0	80	66,66	77,77	72,3
4	68,42	87,5	0	60	66,66	77,77	70,76

Nous avons constaté de grandes similitudes entre ces résultats et ceux de la méthode précédente, mais il y a une différence majeure : La classe 3 assez mal représentée précédemment n'est maintenant plus du tout reconnue pour minNumObj supérieur à 2, mais le taux de succès reste malgré tout assez haut. Là encore, ces classifieurs et tout particulièrement le classifieur avec minNumObj à 3 permettrait de savoir avec une précision assez importante si le verre analysé correspond à une des classes étudié dès lors que l'on a pu exclure la classe 3. Néanmoins, dans l'optique de trouver un classifieur capable de classer la classe 3, nous continuons nos expériences sur un type d'arbre différent.

## Simple CART Tree

L'algorithme Classification And Regression Trees , ou CART est un algorithme récursif qui va séparer l'ensemble E des éléments en 2 sous-ensembles e1 et e2 tel que l'homogénéité de e1 et l'homogénéité de e2 soit tout les deux supérieures à celle de E. L'opération sera ensuite répétée sur chacun des sous ensemble tant que le critère d'homogénéité ne sera pas atteint. Cette fois, nous avons identifié 2 paramètres qui nous ont semblé pertinent. Le minNumObj une fois de plus, mais aussi le nombre de valeurs aléatoires utilisées pour l'initialisation de l'algorithme (ou "seed" selon l'appellation de Weka).

minNumObj	Seed	Clas.1	Clas.2	Clas.3	Clas.5	Clas.6	Clas.7	Succès
2	1,2 et 3	84,21	87,5	20	80	0	77,77	75,38
3	1	84,21	87,5	0	80	0	77,77	73,84
3	2 et 3	89,47	87,5	0	80	33,33	77,77	76,96
4	1,2 et 3	84,21	87,5	0	80	0	77,77	73,84

Là encore, les résultats obtenus présentent un certain intérêt. On constate qu'en plus de la classe 3, une nouvelle classe n'a pas réussi à être classifiée : la classe 6. Paradoxalement, cet échec nous apporte beaucoup, car notre classifieur a un taux de vrai positif étonnement

haut pour l'ensemble des autres classes, allant de 77.77% jusqu'à près de 90% sous certaines conditions. Le paramètre seed jusqu'alors peu utile permet d'affiner un peu les résultats pour avoir un classifieur optimal pour certaines classes.

## Random Tree

La littérature pouvant donner des définitions assez diverse pour le random Tree, nous prendrons la définition donnée par l'outil Weka lui même, c'est à dire qu'un Random Tree de valeur K est un arbre qui se construit en utilisant à chaque noeuds K attributs choisis aléatoirement pour déterminer la condition de séparation des sous ensembles. La valeur K, essentielle fut testée à différentes valeurs

- 1 : La plus petite valeur possible
- 2 : Une valeur qui reste faible mais laissant place à la confrontation de 2 attributs
- 5 : La valeur centrale
- 9 : La valeur maximale

Nous avons pu en tirer les résultats suivant :

Kvalue	Seed	Clas.1	Clas.2	Clas.3	Clas.5	Clas.6	Clas.7	Succès
1	1	68,42	54,16	40	20	66,66	66,66	56,92
1	2	78,94	70,83	20	80	66,66	44,44	66,15
1	3	63,15	70,83	20	60	66,66	55,55	61,53
2	1	68,42	75	40	20	66,66	55,55	63,07
2	2	63,15	83,33	40	40	100	44,44	66,15
2	3	73,68	75	0	40	0	44,44	58,46
5	1	73,68	91,66	20	40	66,66	66,66	72,3
5	2	78,94	70,83	0	40	100	55,55	64,61
5	3	78,94	75	20	80	66,66	77,77	72,3
9	1,2 et 3	73,68	91,66	60	80	66,66	66,66	78,46

La première observation faite concerne la taille des arbres générés. En effet, alors que les précédentes méthodes génèrent des arbres avec une taille n'excédant pas 29 sommets, nous obtenons des arbres dont la taille peut atteindre jusqu'à 75 sommets. Néanmoins, alors que les conditions portent à croire au sur-apprentissage, les résultats obtenus nous indique

le contraire. Pour une KValue faible, le résultat obtenu dépend extrêmement de l'aléatoire, laissant place à un grand nombre d'erreurs. Avec une KValue moyenne, les résultats obtenus sont nettement plus intéressants, pouvant atteindre les 72.3% avec des taux de vrai positif généralement haut pour chaque classe mais, là encore, le taux de réussite de la classe 3 est trop faible pour considérer le classifieur comme étant performant quant à la détection de verres de ce type. Enfin, alors qu'il serait légitime de se dire qu'utiliser systématiquement les 9 paramètres pour la création de l'arbre nous ferait perdre cet effet aléatoire caractéristique du Random Tree, l'arbre obtenu se trouve être l'un des plus pertinent des arbres jusqu'alors générés. Comparativement, le taux de succès général est au-dessus des précédents, atteignant le score de 78.46%, néanmoins aucune classe n'a de taux de vrai positif réellement faible. Ce résultat est d'autant plus surprenant cette précision est atteinte malgré la taille imposante de l'arbre (voir Figure suivante) créer à partir d'un faible jeu de données. Même la classe 3 jusqu'alors difficile à déterminer atteint un taux de vrai positif de 60%, faisant de ce classifieur le meilleur classifieur si l'on essaye de déterminer la nature d'un verre sans avoir à faire un potentiel traitement préalable pour exclure certaines classes.

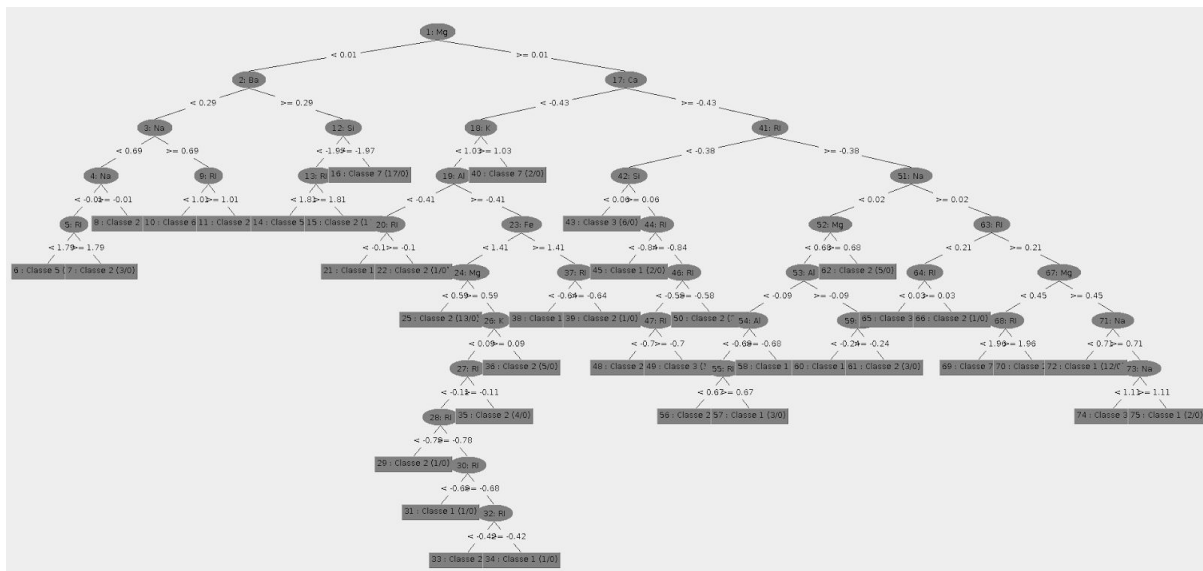


Figure : Aperçu du Random Tree pour KValue = 9

## Random Forest

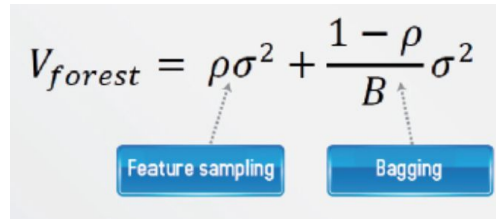
L'idée de base du Random Forest, que partagent l'ensemble des méthodes ensemblistes est assez intuitive : plutôt que d'avoir un estimateur très complexe censé tout classifier à lui seul, on en construit plusieurs de moindre qualité individuelle qui possèdent une vision réduite du problème. Ensuite, on réunit l'ensemble de ces estimateurs pour avoir une vision globale.

Ici, le but sera de créer plusieurs arbres de décisions plus réduits afin d'obtenir une classification. La création de ces arbres est basée sur deux principes :



- le *feature sampling*: chaque sous-arbre créé ne disposera pas de l'ensemble des variables du problème à sa disposition, mais d'un sous-ensemble. Et c'est dessus qu'il devra créer ses règles
- le *tree bagging* : chaque arbre créera une classification sur un sous-ensemble des échantillons d'entraînement uniquement, et non pas la totalité de la base.

Ces dispositions visent à faire baisser la variance du problème.

$$V_{forest} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$


Lors de nos expérimentations, nous avons choisi d'utiliser le critère de Gini comme critère de *split*, car celui-ci tend à donner des arbres de meilleure qualité. Ce critère va tenter de séparer le plus rapidement les classes les plus importantes d'abord, et ce récursivement jusqu'en base de l'arbre.

Parmi les hyperparamètres réglables, nous allons en faire varier 3 qui nous semblent les plus importants

- Le nombre d'arbres votant : nous choisirons 3 valeurs de différentes amplitudes (20, 100, 500)
- Le nombre de caractéristique par arbre : soit la racine carrée du nombre total de variables, soit la totalité des variables
- Le nombre minimal d'exemples par feuille : soit 1 exemple est suffisant pour créer une règle, ou alors 5% du jeu de données

Dans les tableaux suivants, nous notons les différents scores moyens obtenus après une cross-validation (suivant la méthode k-fold, avec k = 3) selon toutes les combinaisons de paramètres possibles.

Nombre variables : sqrt	Voteurs = 20	Voteurs = 100	Voteurs = 500
Sample leaf = 1	0.74	0.72	<b>0.75</b>
Sample leaf = 5%	0.68	0.68	0.69

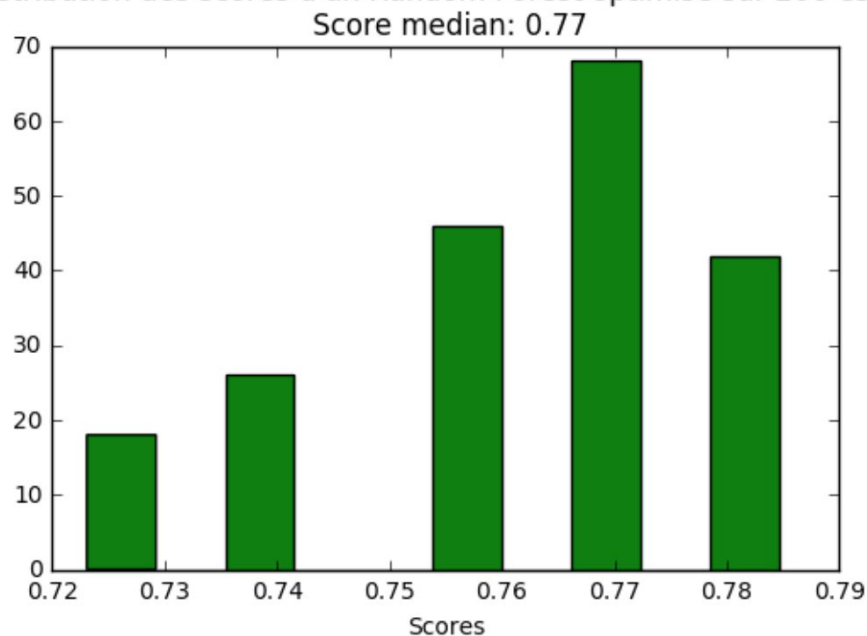
Nombre variables : tout	Voteurs = 20	Voteurs = 100	Voteurs = 500
Sample leaf = 1	0.65	0.72	0.72
Sample leaf = 5%	0.66	0.68	0.69

Les paramètres optimaux pour notre jeu sont donc :

- Nombre de variables :  $\sqrt{n}$ , où  $n$  est le nombre de variables de notre jeu de données (ici, 9)
- Arbres voteurs : 500 estimateurs
- Nombre d'échantillon minimal par feuille : 1

En se servant de ces hyperparamètres, nous faisons 200 essais et observons la distributions des scores afin de déceler une tendance globale dans les scores.

Distribution des scores d'un Random Forest optimisé sur 200 essais



Enfin, en se servant d'un modèle avec lequel on obtient un score médian, nous faisons la classification notre base de test, pour obtenir la matrice de confusion suivante.

	Classes prédites						
		1	2	3	5	6	7
Classes de vérité	1	17	1	1	0	0	0
	2	4	20	0	0	0	0
	3	3	0	2	0	0	0
	5	0	2	0	3	0	0
	6	0	1	0	0	2	0
	7	0	2	0	0	0	7

Nous obtenons donc les taux de succès suivants par classe:

- Classe 1 : 89,5%
- Classe 2 : 83,3%
- Classe 3 : 40,0%
- Classe 5 : 60,0%
- Classe 6 : 66,6%
- Classe 7 : 77,8%

On peut remarquer une corrélation entre le nombre de d'exemples par classes et le taux de succès de classification des exemples de celle-ci. Étant donné que nous ne possédons que très peu d'exemples sur certaines classes, notre classifieur a du mal à généraliser sur les caractéristiques propres que des membres de cette classe pourraient avoir.

Bien que les modèles Random Forest tendent à être plus performants que les arbres de décision, un de leur défaut est le sacrifice d'une partie des capacités explicatives du modèle. Ainsi, nous ne sommes pas totalement en mesure de comprendre comment notre modèle arrive à prendre ces décisions, ce qui rend la retranscription du *savoir* obtenu par le modèle dans la vie réelle difficile.

## V. Outils de collaboration

Afin de partager les données entre tous les membres du groupe , nous avons utilisé le logiciel de gestion de version Github. Nous avons pu y déposer les données d'apprentissage et de généralisation sous différentes formes (brutes, centrées réduites, dans l'intervalle  $[-1, 1]$ ) afin que l'on puisse tester chaque configuration sur les classifieur.

Chaque membre du groupe a pu être renseigné sur les tâches qui lui étaient attribuées grâce à un diagramme de Gantt, formalisé grâce à l'application Ganttter sur Google Drive.

Aussi, pour coordonner la rédaction du rapport nous avons utilisé google docs. Ce document centralisé a permis à chaque membre du groupe d'ajouter sa partie au fur et à mesure de ses expériences, facilitant la vue globale du projet de toute l'équipe.

En plus des séances de projet, nous avons aussi organisé des réunions dans le but de coordonner le travail de chacun des membres . Ces réunions ont eu lieu les 8 et 15 décembre 2016, et ont été décidées grâce à une conversation unique de l'ensemble de la promotion sur l'application WhatsApp, nous permettant de communiquer plus facilement ensemble.

## VI. Conclusion

Sur l'ensemble des classifieurs que nous avons utilisé, il semble que ceux qui sont les mieux capables de dégager des non-linéarités soient les plus efficaces.

Pour commencer, notre premier classifieur (les plus proches voisins), le plus simple, n'obtient qu'un taux de succès maximum de 69,23% avec la distance de Manhattan et la prise en compte des 6 plus proches voisins.

Cet estimateur n'essaie pas de détecter des motifs complexes sous-jacents dans les caractéristiques. À contrario, le perceptron multi-couches est fait de telle manière à détecter ce genre de relations, et avec un pas de gradient de 0.02 et 15 neurones sous la couche cachée, notre taux de succès de classification monte aux environs des 77%.

Nous obtenons un résultat similaire avec le classifieur Random Forest, et culminons avec un Random Tree qui permet d'établir un ensemble de règle stricte qui classifie correctement nos exemples de généralisation avec un de taux de succès d'un peu plus de 78%.

Néanmoins, sur l'ensemble des classifieurs nous pouvons observer une tendance globale : les classes 3, 5 et 6 sont les plus difficiles à classer. En effet, nous ne possédons que très peu d'exemples sur elles, de ce fait, les paramètres chargés de reconnaître ces classes ont tendance à sur-apprendre ce peu d'exemples d'apprentissage, et nos quelques exemples de généralisation ont une plus forte chance d'être mal classé.

## VII. Bibliographie / Webographie

[https://fr.wikipedia.org/wiki/Machine\\_%C3%A0\\_vecteurs\\_de\\_support](https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support)

[https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)

[https://www.cs.utah.edu/~suyash/Dissertation\\_html/node11.html](https://www.cs.utah.edu/~suyash/Dissertation_html/node11.html)

<https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

[https://fr.wikipedia.org/wiki/Perceptron\\_multicouche](https://fr.wikipedia.org/wiki/Perceptron_multicouche)

<http://scikit-learn.org/stable/index.html>

<https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>

*Data Science : Fondamentaux et Études de Cas* par Eric Biernat et Michel Lutz

*Make Your Own Neural Network* par Tariq Rashid

[http://www.academia.edu/19755551/K-Nearest\\_Neighbor\\_Classification\\_for\\_Glass\\_Identification\\_Problem](http://www.academia.edu/19755551/K-Nearest_Neighbor_Classification_for_Glass_Identification_Problem)

<http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomTree.html>

## VIII. Annexe