M1 Informatique – UE ML
Compte rendu des TME 1 à 4

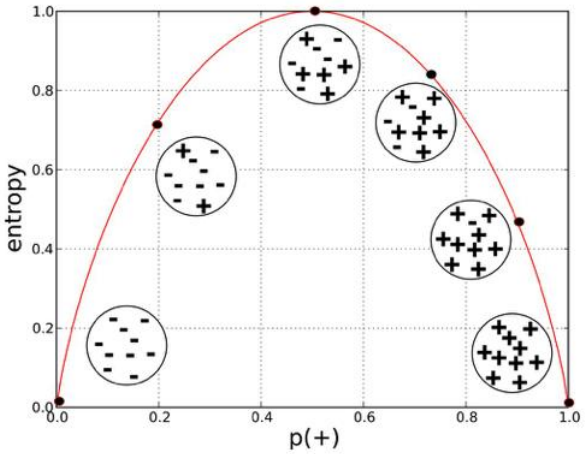
Noms, prénoms:

Jouve Vincent
Cadiou Antoine

TME1 – Arbres de décision

Q 1.3)

L'entropie est une mesure de désordre. Une valeur de 0 dans l'entropie signifie que la partition est complètement ordonnée et ne contiennent que des labels identiques. A l'inverse, si l'entropie est proche de 1 alors la partition est désordonnée car les proportions des labels à l'intérieur tendent à être égales (proche de 1/2)



Le meilleur attribut est donc celui qui a l'entropie conditionnelle la plus faible. Ici, il s'agit de l'attribut 'Drama' avec une valeur de 0.926.

Q1.4)

Nous n'avons pas réussi à utiliser pydot pour visualiser l'arbre, ni même le site graphviz. Nous avons donc essayé de lire dans l'affichage du terminal. Les exemples semblent être divisés en des sous-ensembles de tailles plutôt égales au début (car une faible profondeur ne permet pas de séparer suffisamment les exemples qui sont très désordonnés), mais plus la profondeur augmente, plus les sous-ensembles sont bien ordonnés.

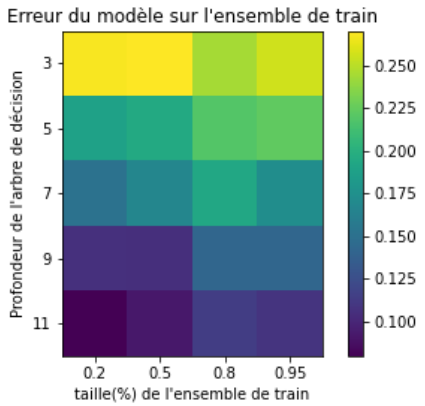
Q1.5)

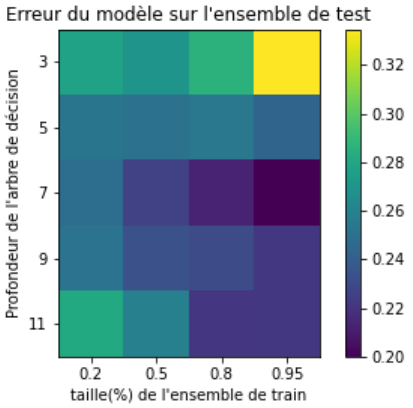
Les scores de bonnes classifications s'améliorent en augmentant la profondeur. En effet, on peut supposer qu'après beaucoup de partitionnements, on se retrouve avec des partitions très petites (de 1 ou 2 éléments par exemple) et où l'entropie de chacune de ces sous-partitions sont de 0.
Il peut s'agir de sur-apprentissage sur les données d'entraînement.

Q1.6)

Comme dit précédemment, il peut s'agir de sur-apprentissage, et l'indicateur peut donc ne pas être pertinent.
Il faut séparer les données en des ensembles de train/test. Ainsi l'indicateur de score obtenu sera plus fiable car plus proche de la réalité.

Q1.7)





(étant donné que les résultats sont en 3D, nous avons opté pour un imshow plutôt que pour des courbes)

Q1.8)
Le comportement n'est pas le même pour les deux erreurs.

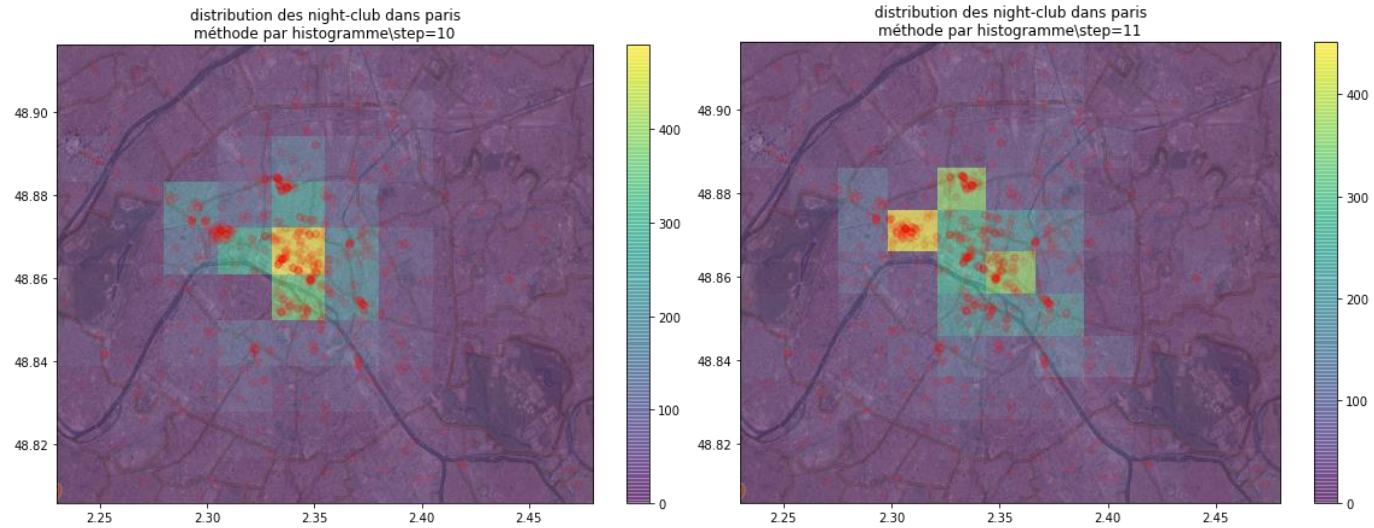
- Sur l'erreur de test (à droite), on remarque que plus il y a d'exemples d'apprentissage, plus l'erreur est faible. Cette erreur diminue également en faisant varier la profondeur. Attention tout de même à ne pas avoir une trop grande profondeur (>7) car l'erreur augmente à nouveau au-delà (car on sur-apprend sur les données d'apprentissage).

- Sur l'erreur d'apprentissage (à gauche), on remarque que le critère le plus impactant sur l'erreur est la profondeur (plus on va profondément, moins l'erreur est haute). Cependant, la taille de l'apprentissage n'influe pas énormément (on a la plus faible erreur avec une taille d'apprentissage de 20%).

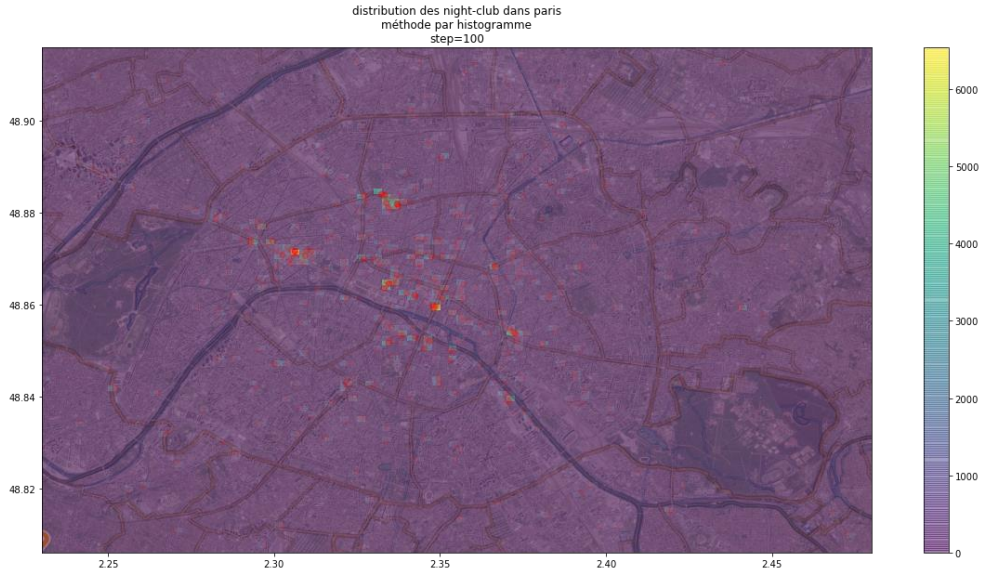
Q1.9)
Les résultats semblent déjà plus fiables et cohérents. Mais ils pourraient être améliorés en procédant par cross-validation.

TME2 – Estimation de densité

Nous nous sommes intéressés au POI ‘night-club’.
Méthode des histogrammes, en faisant varier la taille des zones géographiques :

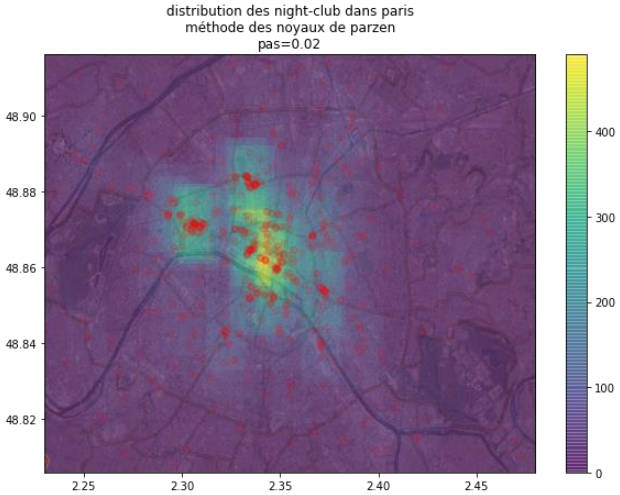


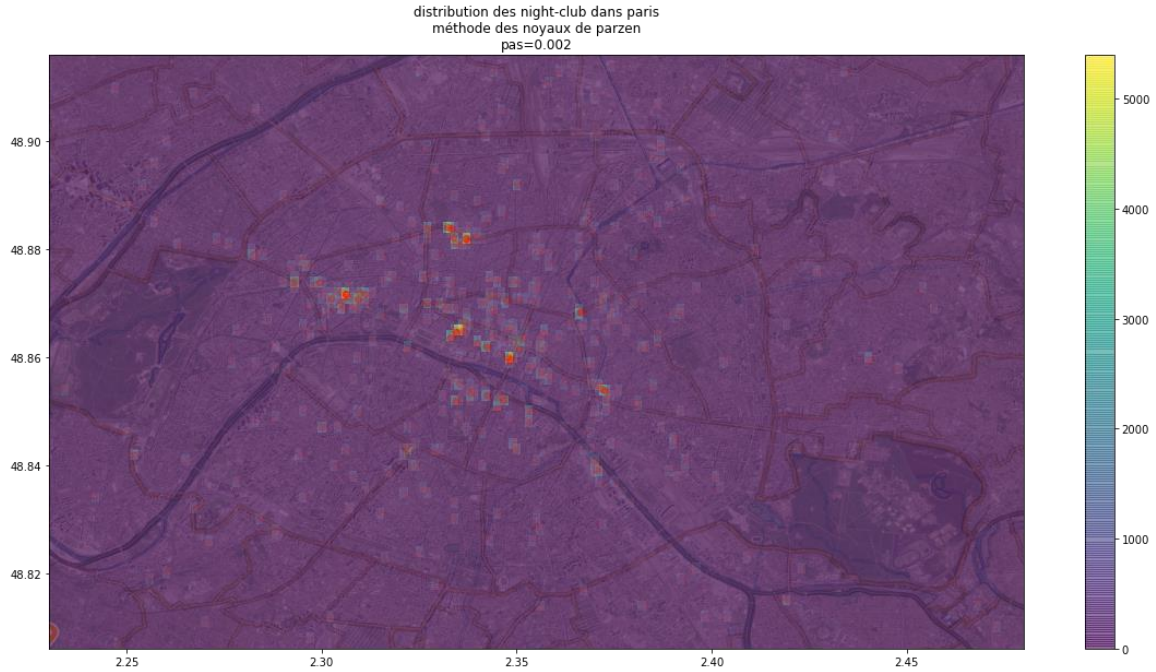
On observe qu’une petite variation dans le grillage implique de grandes variations dans l’estimation de densité



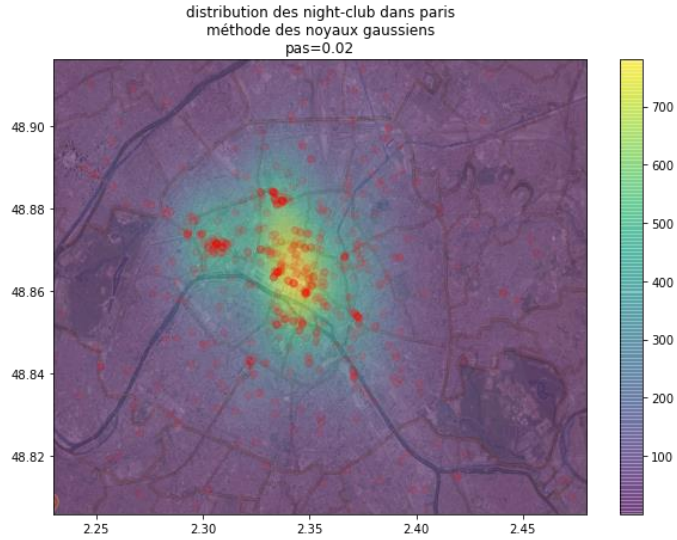
- Ici, avec une step de 100, on colle très bien aux données d’apprentissage, mais cela peut causer des problèmes de sur-apprentissage. Une forte discrétisation favorise donc le sur-apprentissage.

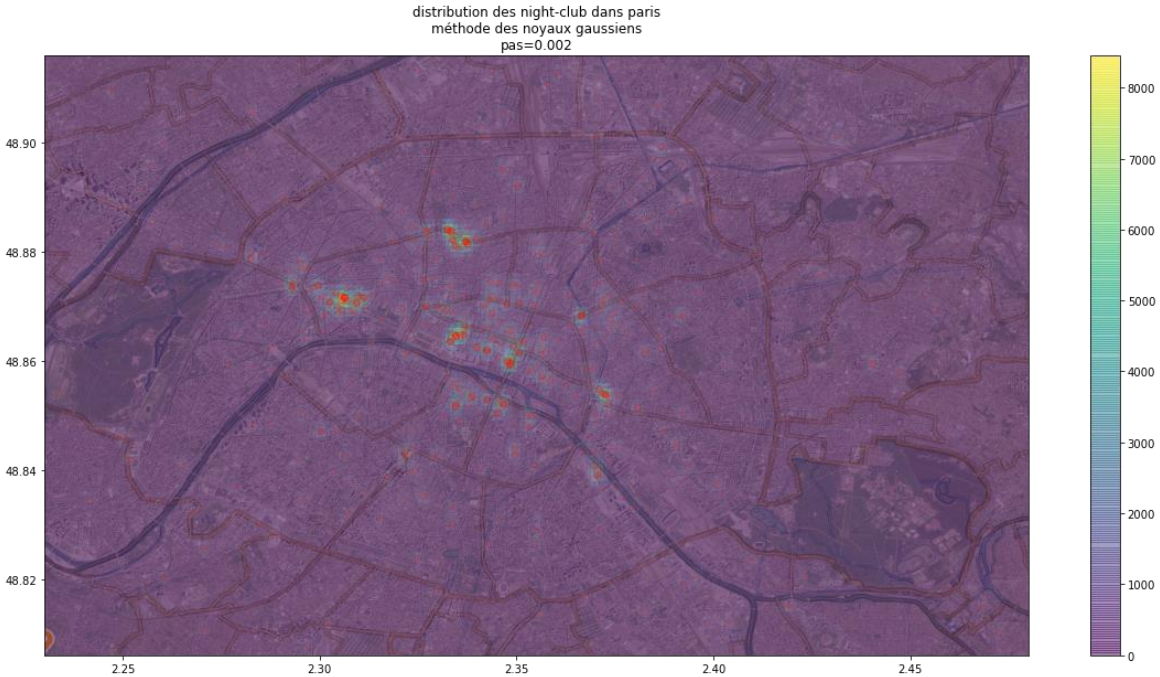
Méthode à noyaux : Parzen





Méthode à noyaux gaussiens (en 2D) :

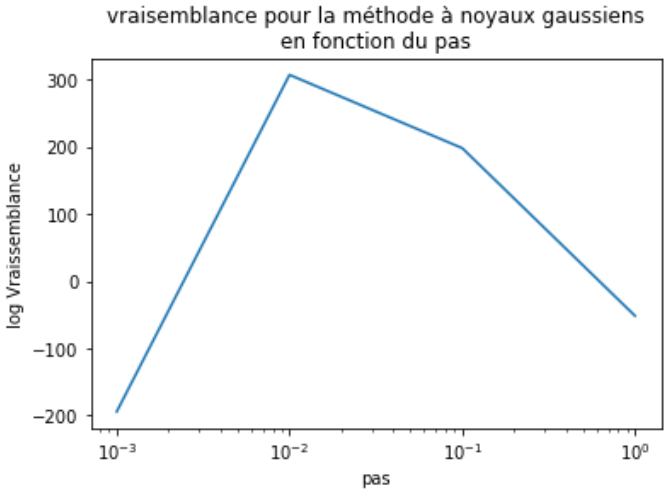




- Le rôle du pas est de définir une zone autour d'un point dans laquelle on calculera sa densité. Plus le pas est grand, moins il y aura de sur-apprentissage. Ainsi, la méthode des noyaux corrige la 'dépendance' à la grille de la méthode des histogrammes.

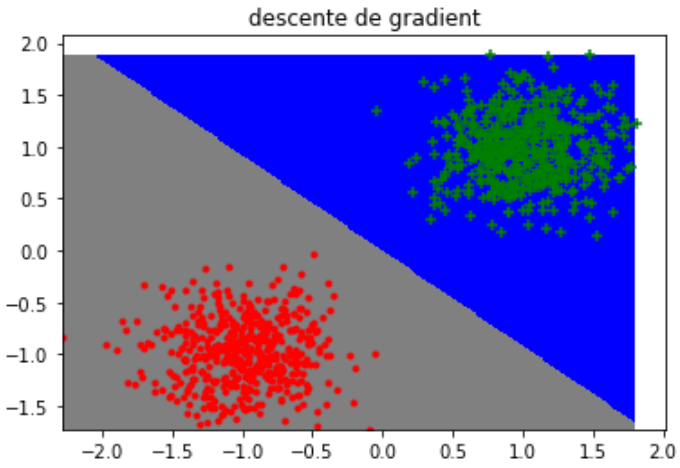
- Les meilleurs paramètres de la méthode à noyaux peuvent se trouver par Grid-Search avec de la cross-validation.

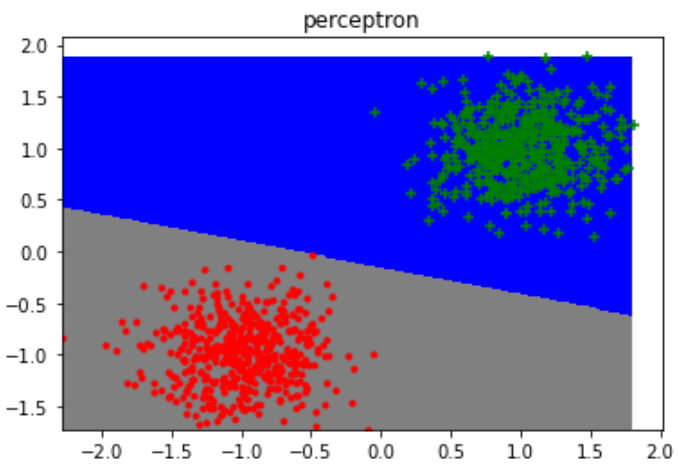
  Ici, nous avons séparé les données en deux ensembles de test(10%) et de train(90%).
  On obtient cette courbe et on voit bien que pour éviter le sur-apprentissage il faut un pas ni trop petit ni trop grand.
  Le meilleur pas semble être 0.01



TME3 – Descente de gradient, Perceptron

Nous avons codé les fonctions de coûts demandées,
Voilà des résultats sur un exemple simple de 2 gaussiennes bien distinctes :

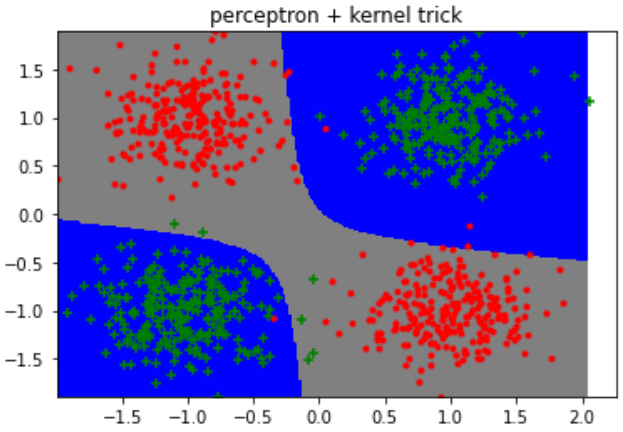




(Au passage, on retrouve bien le 'fameux' problème pour le perceptron dans lequel l'hyperplan séparateur est très « collé » à l'une des deux gaussiennes. Ce problème pourra être résolu plus tard avec les SVM et les marges.)

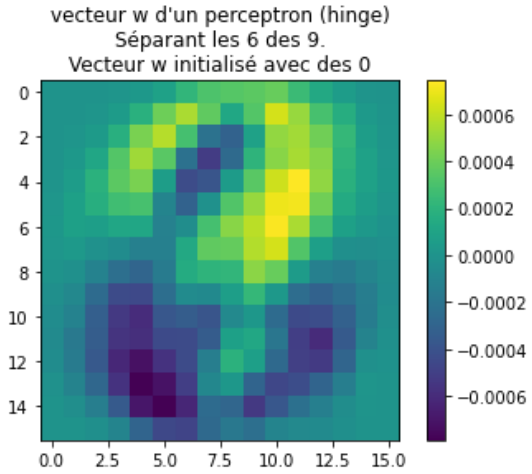
- Nous avons rajouté la prise en compte des biais.
- Nous avons aussi codé les 3 méthodes : batch, mini-batch et stochastique.
  On remarque qu'avec la méthode batch le vecteur w appris est très souvent correct et définit plutôt bien les classes.
  Cependant avec la méthode stochastique il y a de temps en temps des résultats 'improbables', ce qui s'explique à cause du tirage de l'exemple aléatoire de la dernière itération qui peut modifier le vecteur w d'une mauvaise façon.

- Nous avons également codé une version avec un kernel trick pour gérer les cas non linéaires, avec 4 gaussiennes par exemple :

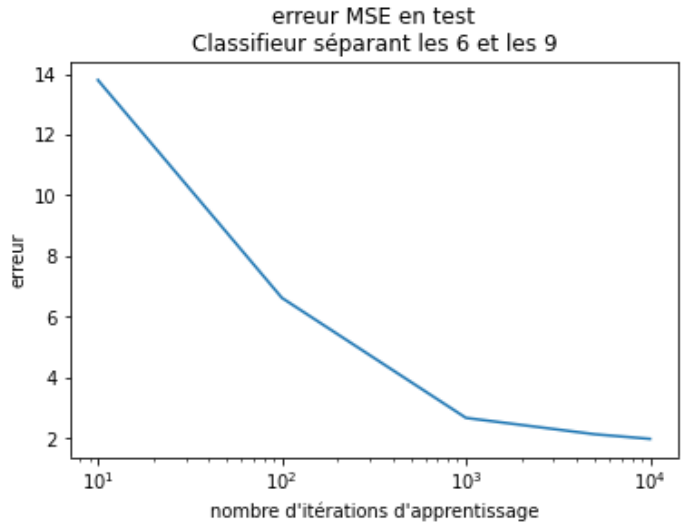


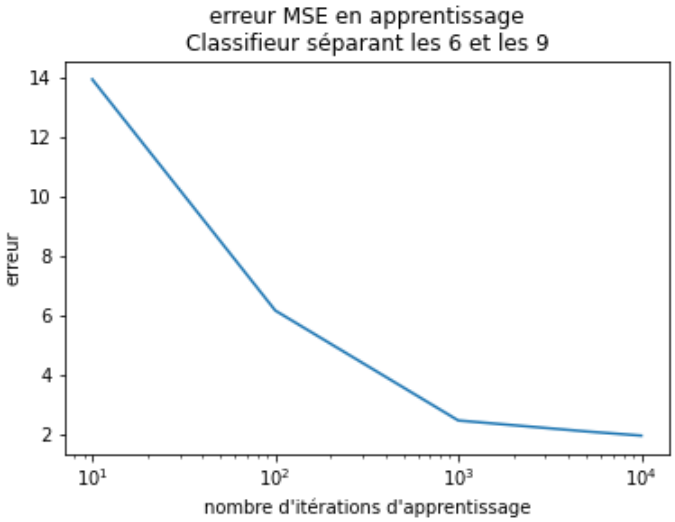
Données USPS :

Voici le vecteur de poids d'un perceptron (vu avec un imshow) permettant de séparer les 6 des 9 :



On visualise ici la matrice de poids w sur un dataset composé de 6 (label -1) et de 9 (label 1). les poids positifs participent pour la classe 9 et les poids negatifs participent pour la classe 6. Ici on voit que la partie arrondie du bas participent plus aux 6
au début au niveau des angles on avait des valeurs positives (qui n'a pas de sens car les pixels sont noirs, il y avait beaucoup de bruit dû à l'initialisation aléatoire du w). Cependant en initialisant le w avec que des 0 on obtient l'image du dessus qui est, je cite, "magnifique". On voit très bien le 6 et le 9





On ne détecte pas spécialement de sur-apprentissage, cependant on remarque que mettre un nombre d'itération trop grand ne sert pas à grand-chose car au-delà de 1000 la variation de l'erreur est minime.
Nous avons utilisé une méthode stochastique pour ces courbes, mais cela aurait été d'autant plus vrai avec une méthode batch ou mini-batch.

TME4 – Perceptron, SVM

Nous allons travailler sur les données MNIST, qui se rapprochent des données USPS du TME3.
Nous allons également appuyer nos résultats grâces aux scores obtenus sur le challenge Kaggle pour le dataset MNIST. https://www.kaggle.com/c/digit-recognizer

Nous avons fait l'apprentissage multi-classe en 'one versus one' et en 'one versus all',
Dans la suite de ce rendu, nous considérerons une classification par 'one versus all'

Le score que nous utilisons est l'accuracy, que nous avons évaluée par cross-validation sur 5 folds.

Classifieur	Accuracy (cross-validation 5 folds)	Score Kaggle (sur des données de tests dont on ne connait pas les labels)
Notre Perceptron	0.8725	0.86642
Perceptron sklearn	0.8720	0.72528

Nous avons ensuite étudié les régularisations Ridge et Lasso

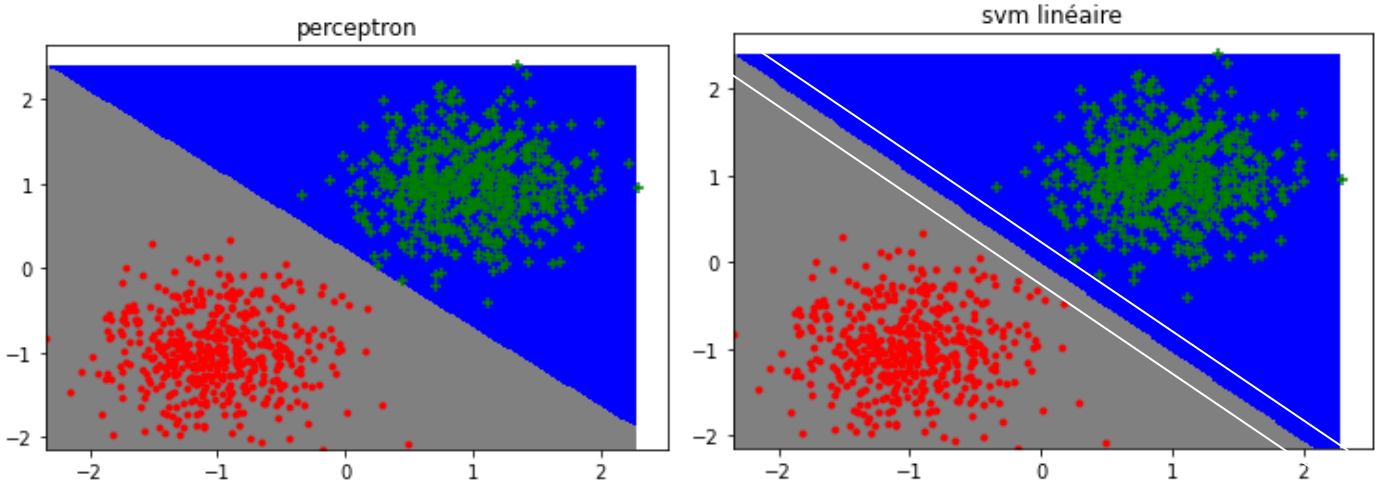
Perceptron sklearn + régularisation ridge (penalty='l2')	0.8422	0.66571
Perceptron sklearn + régularisation lasso (penalty='l1')	0.8663	0.77114

On voit que le Lasso fonctionne un peu mieux que le ridge, cela s'explique probablement car sur une image (28*28) tous les poids ne sont pas forcément pertinents. L'étiquette que l'on veut prédire ne s'explique que par un nombre restreint de variables.

Sur les SVM, nous avons procédé par Grid-Search pour trouver les meilleurs paramètres : kernel, C et gamma.
Kernel = ['poly', 'linear', 'sigmoid', 'rbf'], C=[1, 10, 100, 1000], gamma=[0.1, 0.01, 0.001]
On remarque que, lorsque le kernel est linéaire, les variations de C et de gamma ne changent rien au score.
Nous avons obtenu les meilleurs scores pour un kernel polynomial avec gamma=0.01 et C=100.

SVM kernel='linear'	0.9097	0.86085
SVM kernel='poly', gamma=0.01, C=100	0.9366	0.97228

Voici quelques exemples de séparations sur des jeux de données 2D :
On voit bien pour la SVM linéaire la différence avec le perceptron, à savoir la marge (voir traits blancs)



Regardons les différents noyaux sur un jeu de données avec 4 gaussiennes: