SCIENCES SORBONNE UNIVERSITÉ

Master Données, Apprentissage et Connaissances-DAC

RAPPORT TMES ML

Realisé par :

Liticia TOUZARI
Hanane DJEDDAL
Imad SIDHOUM
Aymen MERROUCHE

Table des matières

1 TME 1 : Arbres de décision, sélection de modèles

L'objectif de ce TME est d'implémenter un classifieur de type Arbre de Décision pour classifier un dataset des films.

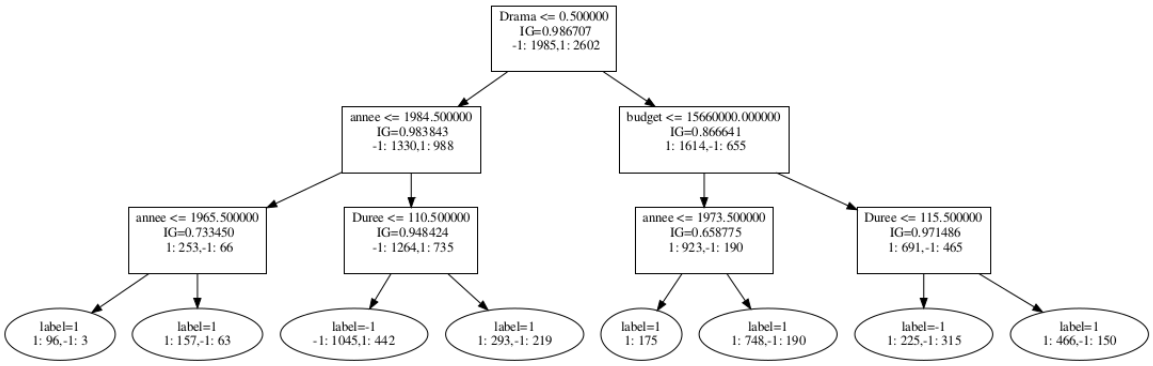


FIGURE 1 – Arbre de Decision de profondeur 3

1.1 Score de bonnes classifications

On calcule le score de bonnes classifications en fonction de la profondeur :

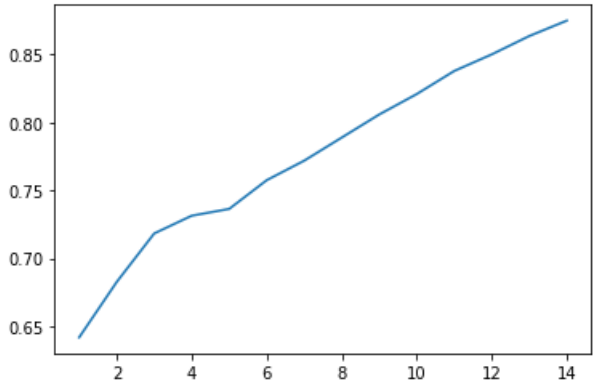


FIGURE 2 – Score de bonnes classifications en fonction de profondeur

En augmentant la profondeur, on obtient des critères plus précis et donc une meilleure classification. Mais cela n'implique pas des bons résultats pour des nouvelles entrées, ce qu'on appele le sur-apprentissage Un indicateur plus fiable sera la bonne classification pour des données test.

1.2 Score Avec des ensembles Train/Test

On divise le dataset en ensembles apprentissage/test. Pour différentes partions, on évalue le score :

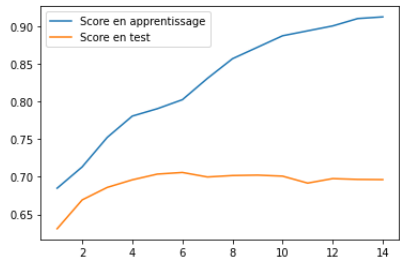


FIGURE 3 – 0.2 pour le train et 0.8 pour le test

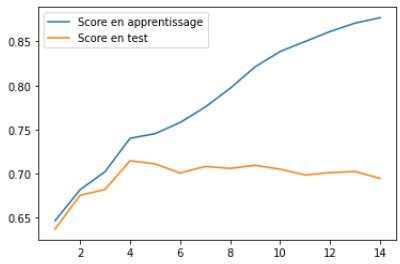


FIGURE 4 – 0.5 pour le train et 0.5 pour le test

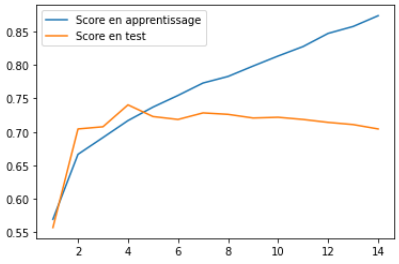


FIGURE 5 – 0.8 pour le train et 0.2 pour le test

On remarque que le score du dataset Test est meilleur quand on assigne plus de données au apprentissage qu'au test.

1.3 Validation Croisée

On divise l'ensemble de données en n sous-ensembles (folds), à chaque itération on teste sur un ensemble et on fait l'apprentissage sur le rest.
Pour n = 3, on obtient l'erreur par ensemble et l'erreur moyenne suivantes :
, 0.7115, 0.7024

0.71026

2 TME 2 : Estimation de densité

L'objectif de ce TME est de prendre en main les algorithmes d'estimation de densité. On choisit un type de POI à traiter et on estime la loi de densité p(x, y) géographique des POIs sur Paris (x, y sont les coordonnées GPS d'un lieu).

2.1 Méthode des histogrammes

L'estimation discrète est alors : $P(x_0 \in POI_j) = \sum_{i=1}^{N} 1_{x_i \in POI_j}$ avec$x_0 = (x, y)$.
Exemple sur type poi = "night club"

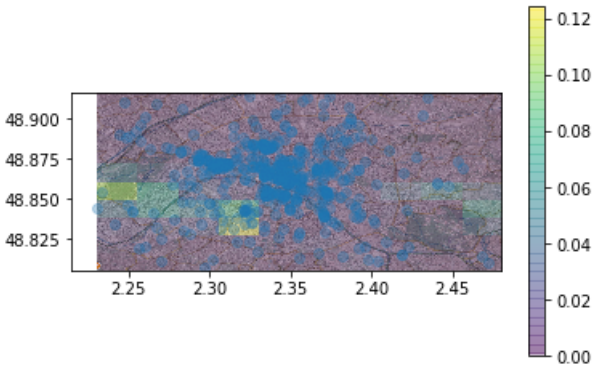


FIGURE 6 – Histogramme avec faible discrétisation

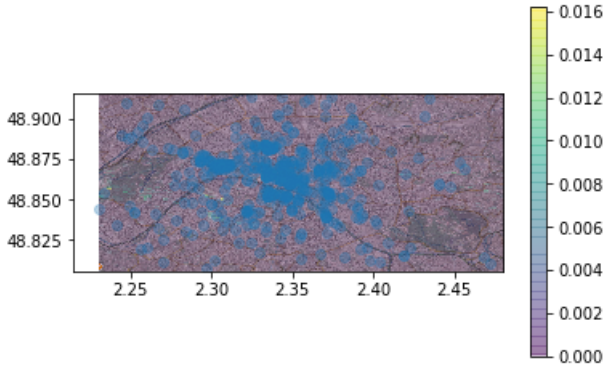


FIGURE 7 – Histogramme avec forte discrétisation

On constate que pour une forte discrétisation qu'il y a beaucoup de cases sans aucun échantillon et celles qui en ont en ont un nombre faible donc peu représentatives.

2.2 Méthode à noyaux

2.2.1 Fenêtre de Parzen

Pour un point x_0, $p_n(x_0) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V_n}\phi(\frac{x_0-x_i}{h_n})$, $\phi(\frac{x_0-x_i}{h_n})$ =1 ssi x est dans hypercube de volume V_n centré en x_0.

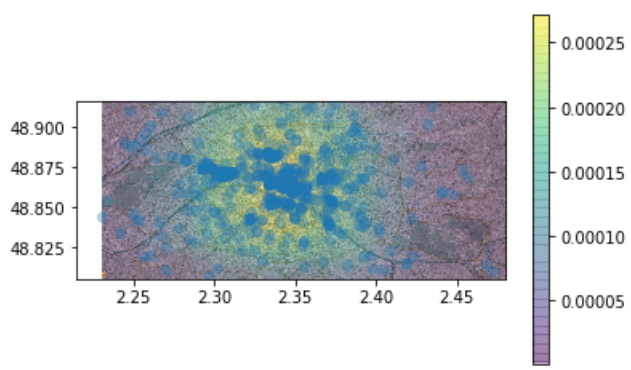


FIGURE 8 – Fenêtre de Parzen avec rayon=0.05

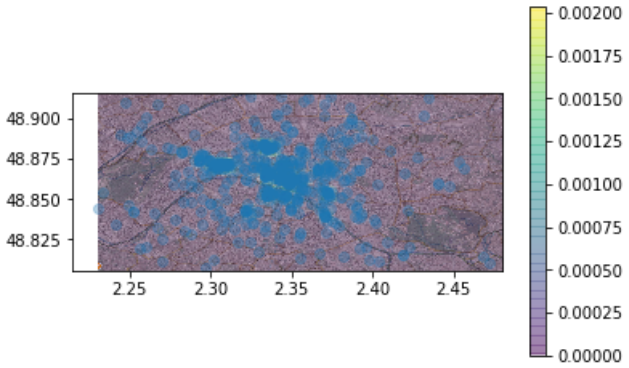


FIGURE 9 – Fenêtre de Parzen avec rayon=0.005

Plus le rayon diminue plus on a de cases vides et non représentatives. Et pour un rayon très grand le modèle est peu sensible. Compromis entre petite résolution et grande variabilité.

2.2.2 Noyau gaussien

ϕ peut être plus générale et donc permettre de pondérer différemment selon la distance au point d'estimation. Pour ce qui suit on opte pour le calcul de distance pour une gaussienne.

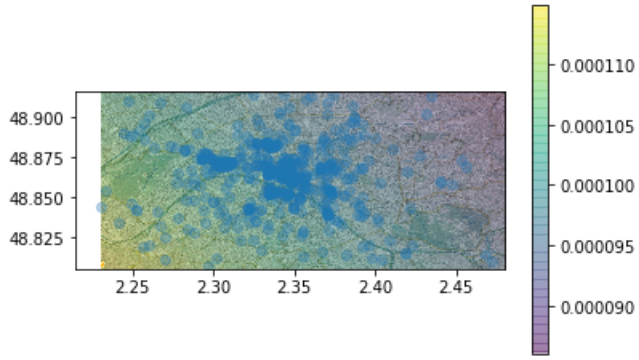


FIGURE 10 – Noyau gaussien avec rayon=0.05

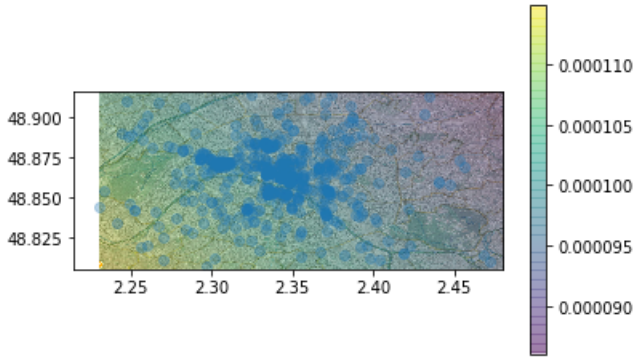


FIGURE 11 – Noyau gaussien avec rayon=0.005

Pour choisir les meilleurs paramètres pour les modèles, on peut évaluer par maximum de vraisemblance et ainsi choisir le meilleur rayon pour le noyau ou la discrétisation adéquate.

Histogramme : rapide, mais coûteuse en mémoire, effet de bords.
Par noyaux : lent, plus flexible, sensibilité de la taille du noyau.

2.3 Classification

Dans cette section, on s'interesse à prédire la note d'un POI en fonction de son emplacement. La matrice X est donc les coordonnées géographiques et le vecteur Y est la note.

2.3.1 KNN

On applique le KNN en considérant la moyenne des notes de K plus proche voisins.
$p(x|y) = \frac{1}{k} \sum_{j, j \in \{kplusproche\}} y_j$

2.3.2 Nadaraya-Watson

L'estimateur Nadaraya-Watson peut être vu comme une moyenne pondérée des Y_i avec un poids $\{W_i(x)\}_{i=1}^n$. Ces poids dépend de x donc l'estimateur est une moyenne locale des Y_i au tour de X=x.
$m(x,h) = \sum_{i=1}^n W_i(x) * Y_i$
avec $W_i(x) = \frac{K_h(x-X_i)}{\sum_{j=1}^n K_h(x-X_j)}$

On évalue le modèle en fonction de K (pour KNN) et h (le paramètre du kernel pour Nadaraya-Watson) pour l'erreur moyenne absolue et l'erreur max :

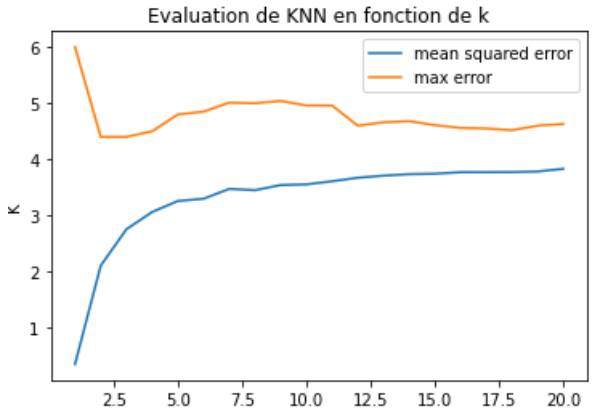


FIGURE 12 – L'erreur de KNN en fonction de k

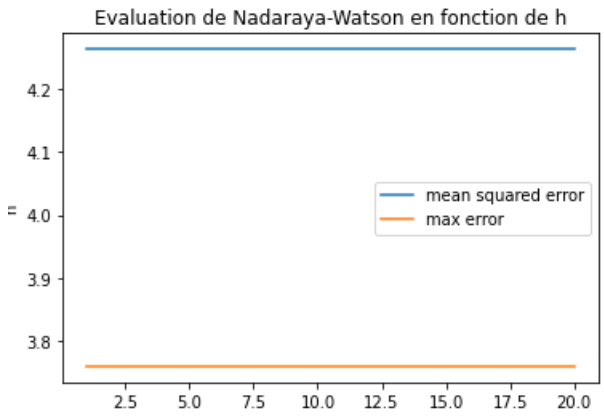


FIGURE 13 – L'erreur de N-W en fonction de h

Le KNN donne une erreur nulle pour K=1 mais cela est un sur-apprentissage (pour un x prendre sa note exact). À partir de k=5 on obtient une erreur presque constante au tour de 3.5 qui est mieux que le résultat de Nadaraya-Watson.
Ces résultats restent insatisfaisants, et on peut conclure que l'emplacement n'as pas une grande influence sur la notation.

3 TME 3 : Descente de gradient, Perceptron

3.1 Perceptron

Les fonctions de calcul d'erreur étudiées dans ce TME sont :
Erreur Hinge(x, y, w) = max(0, y $<$ x.w $>$)
Erreur MSE(x, y, w) = $(< x.w > -y)_2$

3.1.1 Calcul du coût

On teste ces fonctions sur un exemple 2D à l'aide de la fonction plot-error :

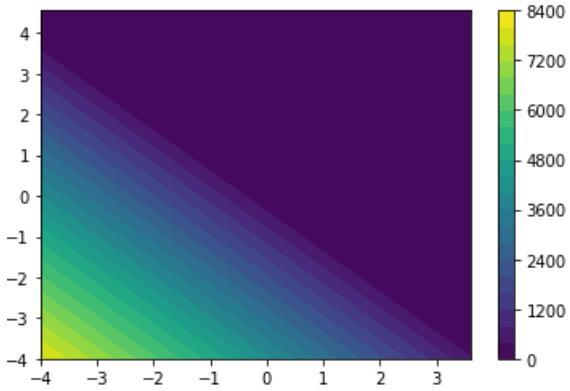


FIGURE 14 – Hinge loss

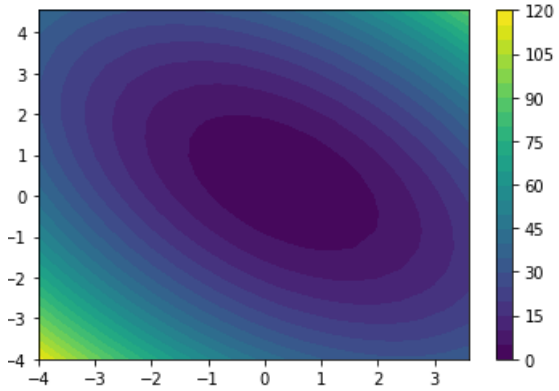


FIGURE 15 – Mse loss

3.1.2 Perceptron

Le perceptron est un algorithme d'apprentissage supervisé de classifieurs binaires (séparant deux classes). Il s'agit alors d'un type de classifieur linéaire.
La fonction d'erreur hinge est utilisé pour le calcul de l'erreur, le plot suivant représente l'évolution du tracé de la frontière :

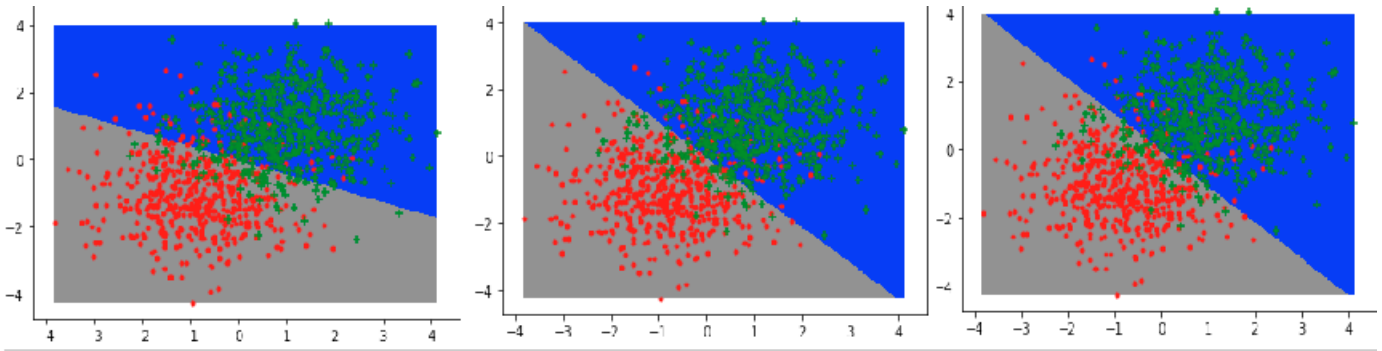


FIGURE 16 – Frontière de décision du perceptron

3.1.3 Régression Linéaire

En général, le modèle de régression linéaire désigne un modèle dans lequel l'espérance conditionnelle de y sachant x est une transformation affine de x. Cependant, on peut aussi considérer des modèles dans lesquels c'est la médiane conditionnelle de y sachant x ou n'importe quel quantile de la distribution de y sachant x qui est une transformation affine de x.
Le modèle de régression linéaire est souvent estimé par la méthode des moindres carrés.
Pour notre exemple, la fonction d'erreur mse est utilisée pour le calcul de l'erreur, le plot suivant représente l'évolution du tracé de la frontière :

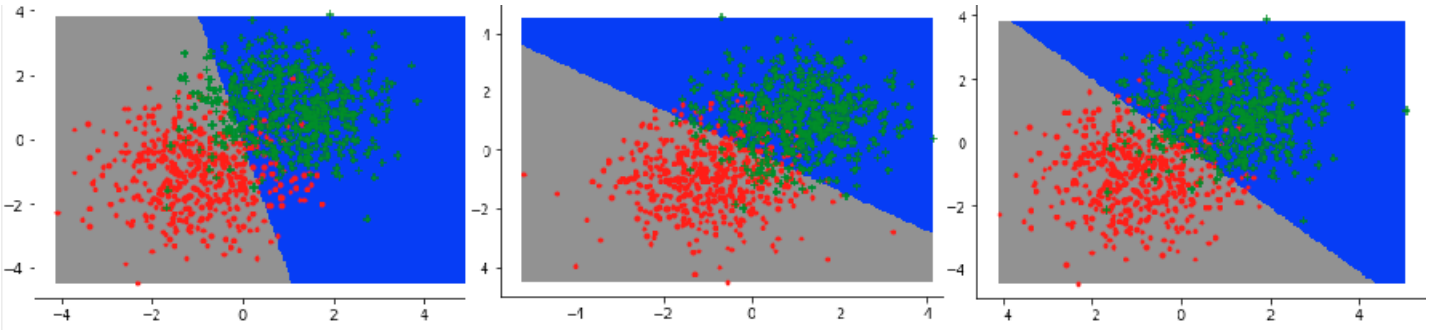


FIGURE 17 – Frontière de décision pour la régression linéaire

3.1.4 Perceptron avec biais

On calcule la somme de tous les points pondérés $\sum_{i=1}^{2} w_i \cdot x_i$ et on ajoute un certain biais b. Ce biais peut être vu comme un neurone externe supplémentaire qui envoie systématiquement le signal 1 de poids b. Grâce à lui, la frontière va être décalée et le réseau aura donc de plus grandes opportunités apprentissage.

Score pour perceptron sans biais : données train : 0.910, données test : 0.912
Score pour perceptron avec biais : données train : 0.909, données test : 0.913

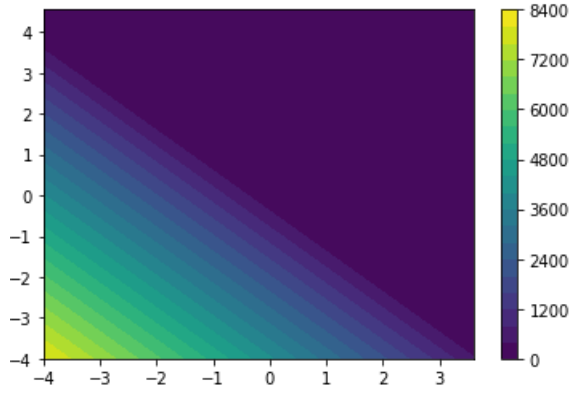


FIGURE 18 – Perceptron sans biais

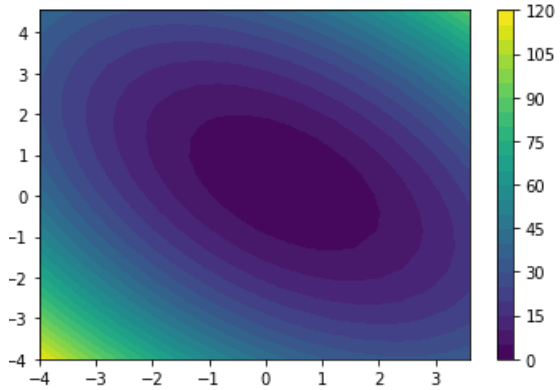


FIGURE 19 – Perceptron avec biais

3.1.5 Bonus : Descente de gradient stochastique et mini-batch

Dans le cas de très grands ensembles de données, l'utilisation de Descente de gradient peut être assez coûteuse car nous ne faisons qu'une seule correction pour un passage sur l'ensemble d'entraînement. Ainsi, plus l'ensemble d'apprentissage est grand, plus notre algorithme met à jour les poids lentement et plus il peut prendre du temps jusqu'à ce qu'il converge vers le coût minimum global. Dans la descente de gradient stochastique, nous n'accumulons pas les mises à jour de poids, w est amélioré pour chaque x parcouru.

Le Mini-Batch est un compromis entre une Descente de Gradient simple et le Gradient Stochastique. Dans Mini-Batch, nous mettons à jour le modèle en fonction de petits groupes d'échantillons d'apprentissage ; au lieu de calculer le gradient à partir de 1 échantillon ou de tous les n échantillons d'apprentissage, nous calculons le gradient à partir de 1 <k <n échantillons d'apprentissage. Mini-Batch converge en moins d'itérations qu'une déscente de Gradient car nous mettons à jour les poids plus fréquemment ; cependant, Mini-Batch nous permet d'utiliser un fonctionnement vectorisé, ce qui se traduit généralement par un gain de performances de calcul par rapport au Gradient Stochastique.

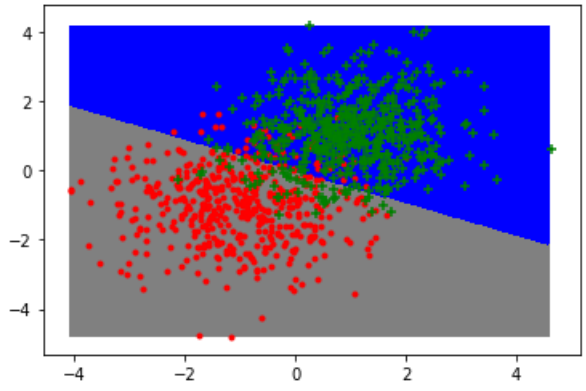


FIGURE 20 – Gradient stochastique

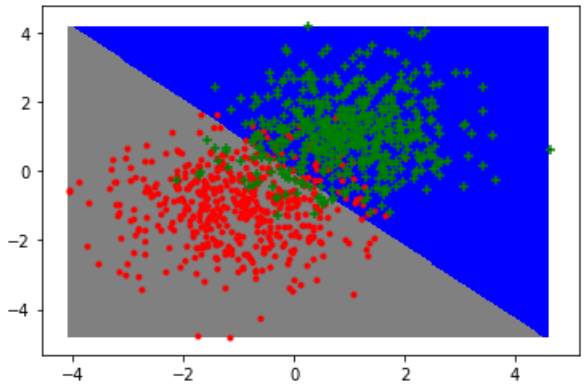


FIGURE 21 – Gradient mini-batch

3.2 Données USPS

Teste des algorithmes sur les données USPS :

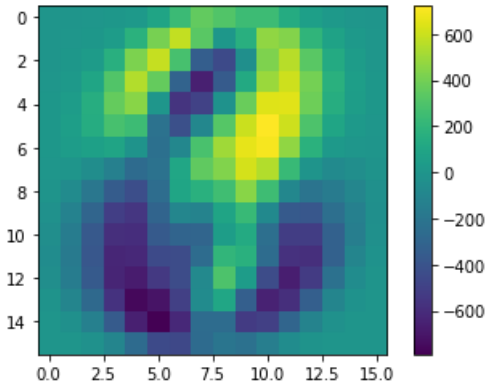


FIGURE 22 – Matrice de poids classe 6 vs classe 9 avec Perceptron

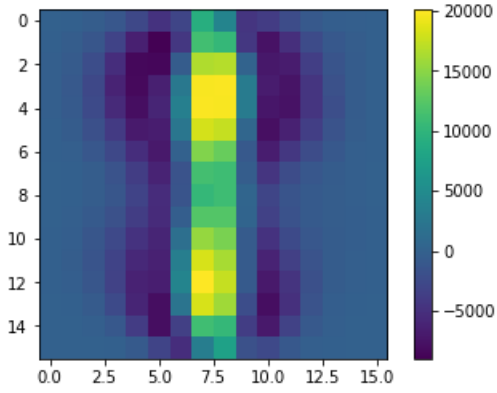


FIGURE 23 – Matrice de poids classe 1 vs classe 8 avec Perceptron

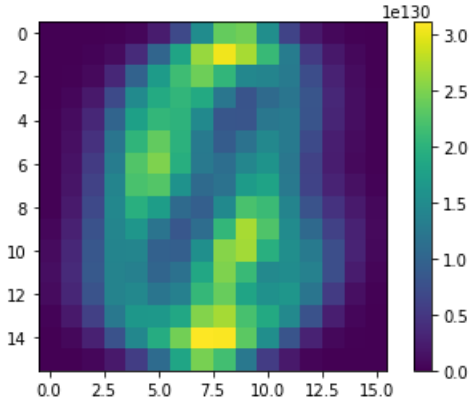


FIGURE 24 – Matrice de poids classe 6 vs classe 9 avec Régression Linéaire

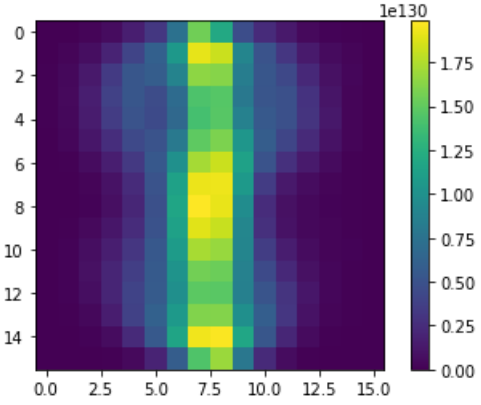
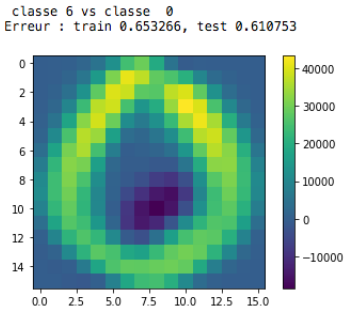
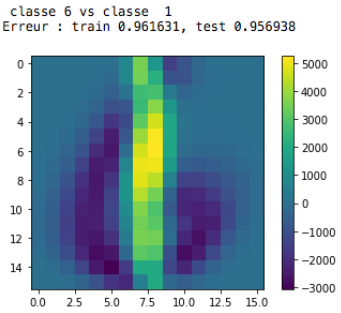
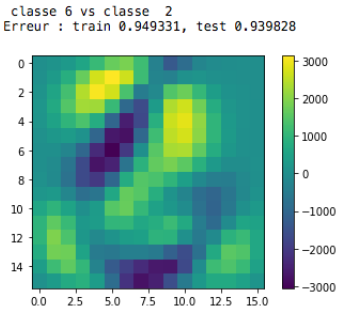


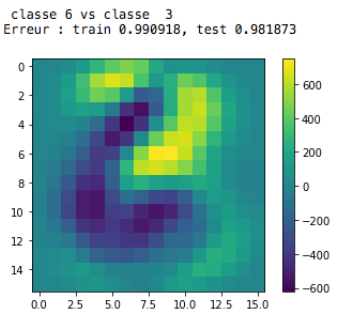
FIGURE 25 – Matrice de poids classe 1 vs classe 8 avec Régression Linéaire

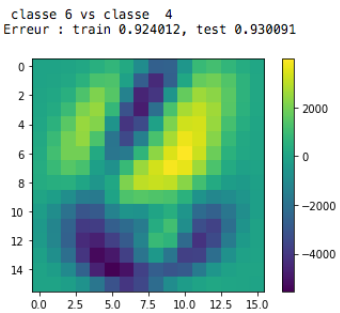
Classe 6 contre les autres classes :

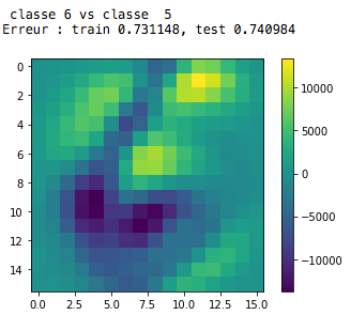


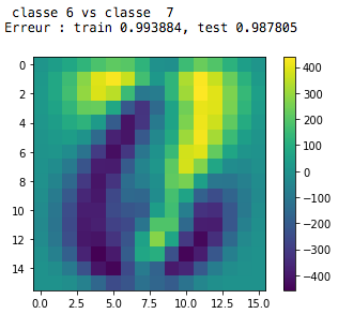


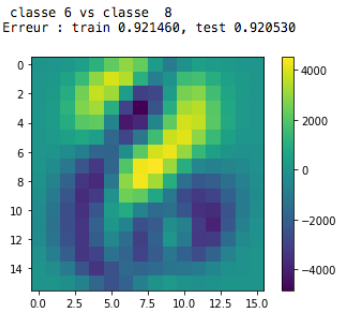


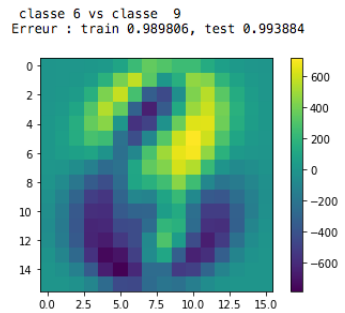












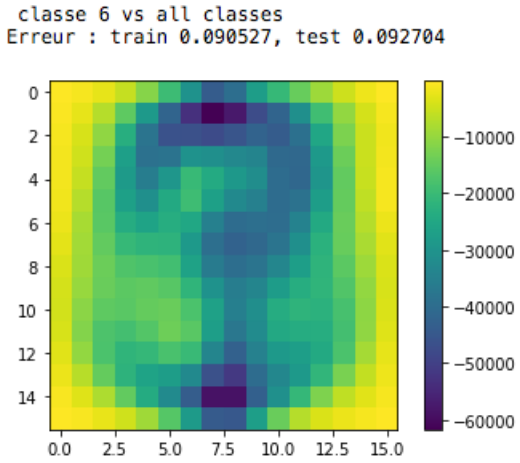


FIGURE 26 – Perceptron classe 6 vs All

Tracés des courbes d'erreurs en apprentissage et en test en fonction du nombre d'itérations :

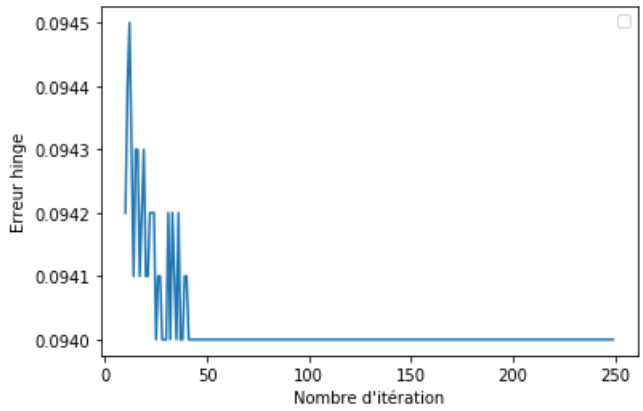


FIGURE 27 – Variation de l'erreur hinge en fonction du nombre d'itération sur les données train

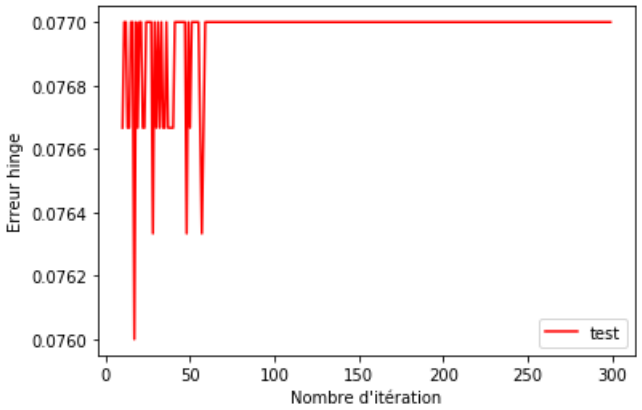


FIGURE 28 – Variation de l'erreur hinge en fonction du nombre d'itération sur les données test

DONNÉES TRAIN : On constate que l'erreur en générale diminue qu'à partir de 50 itérations ou elle se stabilise.
DONNÉES TEST : L'erreur diminue, puis augmente à partir de la valeur 70 itérations ce qui est dû à un sur-apprentissage lors de l'augmentation des nombres d'itérations.

4 TME 4 : SVM

Le SVM (support vector machine) est un modèle linéaire basé sur la recherche de l'hyperplan de marge optimal et il a été prouvé qu'il donne de meilleurs résultats pour les données séparables linéairement. Dans ce TME, nous allons explorer les outils scikit-learn et expérimenter avec les données déjà utilisées dans les TMEs précédents afin de comparer les résultats avec les résultats des classifieurs déjà mis en œuvre.

4.1 Introduction : Module scikit-learn

- Comparaison entre le perceptron implémenté dans TME3 et celui de scikit-learn

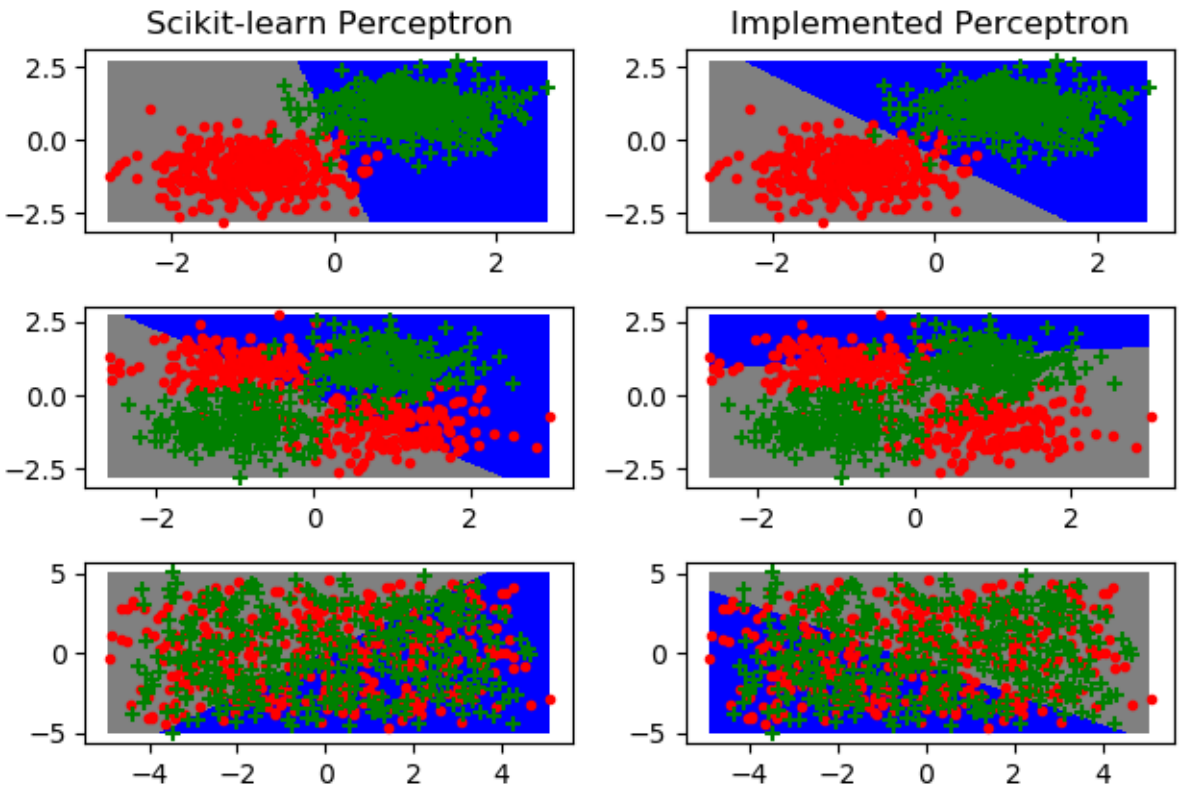


FIGURE 29 – Comparaison entre les deux perceptron pour les différents types de données

Modèle	Perceptron de scikit-learn			Perceptron implémenté		
Type de données	0	1	2	0	1	2
Erreur en train	0.016	0.53	0.50	0.028	0.51	0.47
Erreur en test	0.017	0.52	0.52	0.031	0.48	0.51

On remarque que le perceptron de scikit-learn classifie un peu mieux que le nôtre lorsque les données sont linéairement séparables. Sinon, pour les autres types de données, les deux classifieurs sont assez similaires.

4.2 SVM et Grid Search

4.2.1 SVM et expérimentations

Dans ce qui suit nous allons tester l'algorithme SVM sur des données artificielles 2D, et ce, pour plusieurs noyaux : linéaire, polynomiale et gaussien. Nous allons aussi tester plusieurs paramétrages de ces noyaux, et étudier les frontières de décision ainsi que le nombre de vecteurs supports utilisé pour l'apprentissage de chaque modèle.

Noyau linéaire :

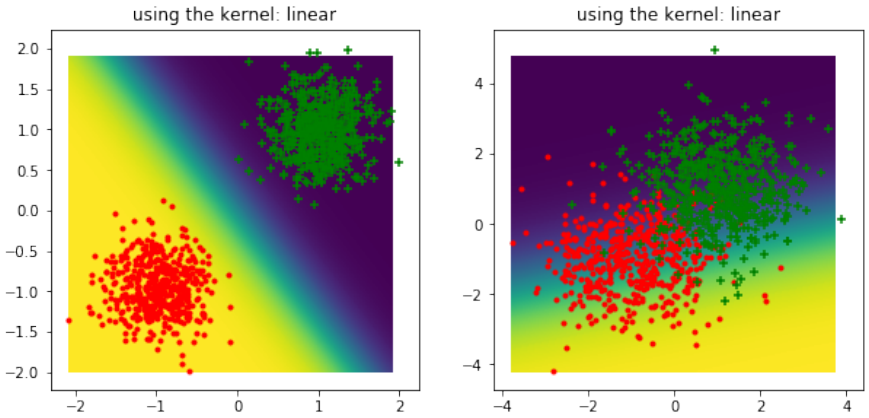


FIGURE 30 – Classification SVM sur mélange de deux gaussiennes | noyau linéaire

On teste l'algorithme SVM, sur des données 2d artificielles (mélange de deux gaussienes). Dans la figure de gauche, le train set n'est pas bruité, les données sont séparables linéairement. La frontière de décision est franche. L'erreur de classification sur le test est de 0. Sur l'image de droite, les données sont bruitées, l'erreur de classification sur le test est de 0.21.

Les vecteurs supports :

Dans l'algorithme SVM, le vecteur de poids appris est une combinaison linéaire des exemples d'apprentissage : $\boldsymbol{w} = \sum_i \alpha_i \times y^i \times \boldsymbol{x^i}$. En réalité très peu d'exemples sont utilisés pour trouver la frontière de décision que définit $\boldsymbol{w}$, ce sont les seuls exemples dont les α_i correspondants, dans la formule précédente, sont supérieurs à 0. Il s'agit des points qui se trouvent sur la marge (Dans l'algorithme SVM et pour plus de robustesse, la frontière de décision n'est pas collée aux données. Elle est centrée par rapport aux différents groupes homogènes. Ces groupes sont à équidistance de la frontière de décision, c'est cet espace qu'on appelle marge. Il contient les points qui sont proches de la frontière de décision, points difficiles à classer). Ce sont les points discriminants sur lesquels se basent la construction du (w). Ils sont appelés vecteurs supports. Les autres points faciles à classer (qui se situent dans un groupe de points homogène) ne sont pas utilisés et donc les α_i qui leur correspondent sont nuls. Ces exemples n'apportent pas d'information pour la construction de la frontière. Plus la frontière de décision de l'algorithme est complexe plus le nombre de vecteurs supports augmente.

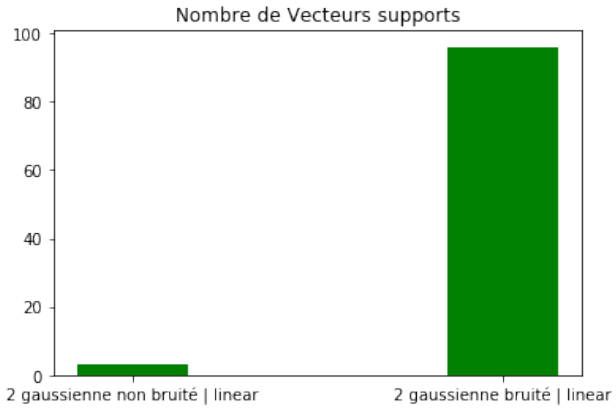


FIGURE 31 – Nombre de vecteurs supports | sur données bruitées et non bruitées | noyau linéaire

La figure représente un histogramme des nombres de vecteurs supports pour les deux exemples précédents i.e classification SVM sur une mixture de deux gaussiennes, l'une sans bruit et l'autre avec bruit. Dans le premier exemple, la frontière de décision est franche. SVM n'a besoin que de deux vecteurs supports pour construire le $\boldsymbol{w}$: un point de chaque classe. Dans le deuxième cas, la frontière de décision est complexe, de nombreux exemples se trouvent dans la marge. SVM dans ce cas a besoin de 95 vecteurs supports pour construire $\boldsymbol{w}$

Données non séparables linéairement :

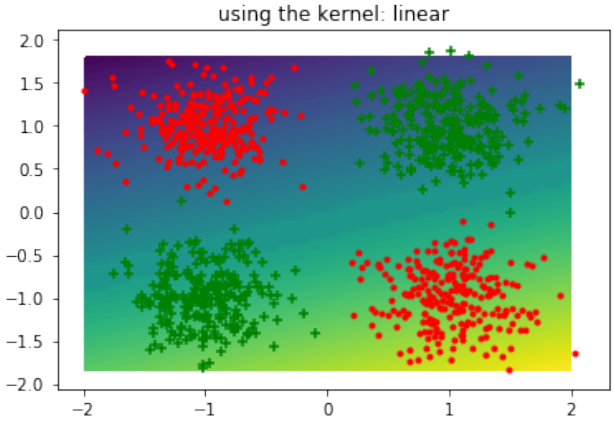


FIGURE 32 – Classification SVM | quatre gaussiennes | sans bruit

La figure représente le résultat de la classification SVM avec noyau linéaire sur un jeu de données 2D non-séparable linéairement (mixture de quatre gaussiennes). Il est impossible de séparer les deux classes dans ce cas avec une droite. L'erreur de classification sur le train est de 0.5 autant dire qu'il s'agit d'un classifieur aléatoire.

L'algorithme SVM utilisé jusqu'à présent ne peut donc résoudre que les problèmes séparables linéairement, ce qui est trés rarement le cas dans des applications réelles. Pour palier à ce problème on introduit l'astuce du noyau "the kernel trick" qui permet de projeter les exemples sur un espace de plus grande dimension et ainsi accroître leur expressivité. Cette astuce permet d'obtenir des frontières de décision non-linéaires.

Noyau Polynomiale :

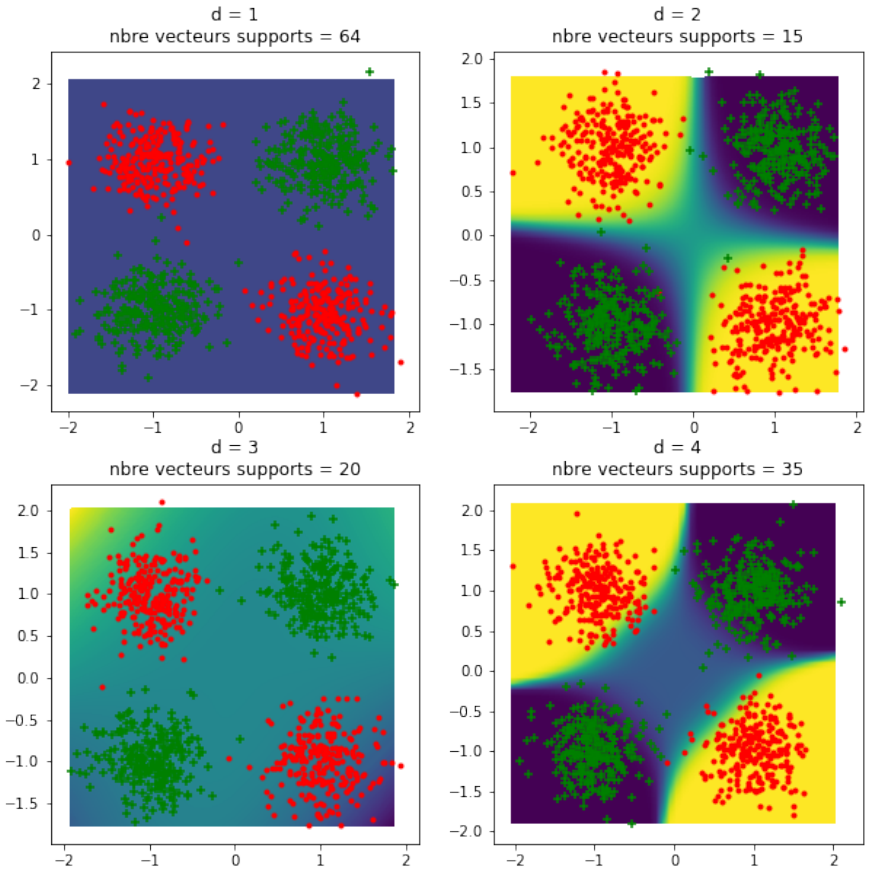


FIGURE 33 – Classification SVM sur données non séparables | noyau polynomiale | en variant le degré

On teste le noyau polynomial sur les quatre gaussiennnes. Le noyau polynomial prend en paramètre le degré du polynôme, par exemple quadratique pour d=2. Plus le d est grand plus la frontière de décision est flexible. Un degré égal à 1 revient au modèle linéaire. Dans ce cas le degré qui marche le mieux est le degré 2, avec une erreur de $0,001$ sur le test. Quant au nombre de vecteurs supports, plus la frontière est complexe plus l'algorithme en requiert plus. La meilleure frontière obtenue pour $d = 2$, requiert seulement 15 vecteurs supports, la coupe est nette, la marge contient peu d'exemples.

Noyau "rbf" radial basis function kernel :

Prenons un exemple beaucoup plus difficile : un échiquier. Le kernel polynomial ne marche pas du tout, avec une erreur de $0,5$: classifieur aléatoire. On teste donc le kernel rbf : radial basis function kernel.

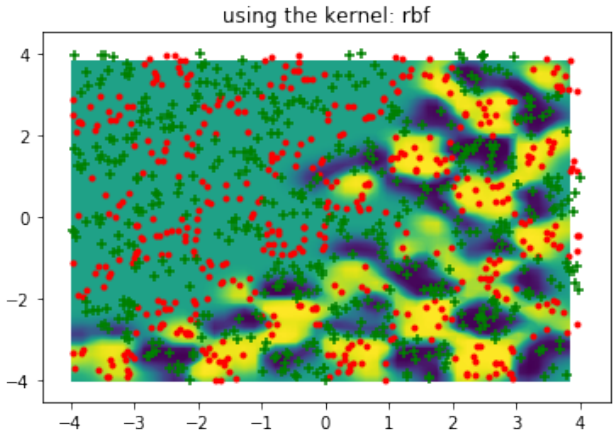


FIGURE 34 – Classification SVM sur données en échiquier | noyau rbf

On obtient une erreur de 0.28 sur le test avec le kernel rbf.

Ce modèle dépend d'un paramètre : γ. Plus γ est grand plus le modèle essaie d'apprendre exactement les données, c'est-à-dire qu'on accroît l'importance qu'on attribut à chaque exemple du train. Un autre paramètre que nous allons tester et dont dépendent d'ailleurs tous les modèles SVM est le C. Plus le C est grand plus on favorise l'accuracy à la taille de la marge (on favorise l'accuracy au détriment de la robustesse). Pour les petites valeurs de C, on obtient une large marge, donc une frontière plus robuste, mais au prix de l'accuracy (ce paramètre permet de réguler le sur apprentissage).

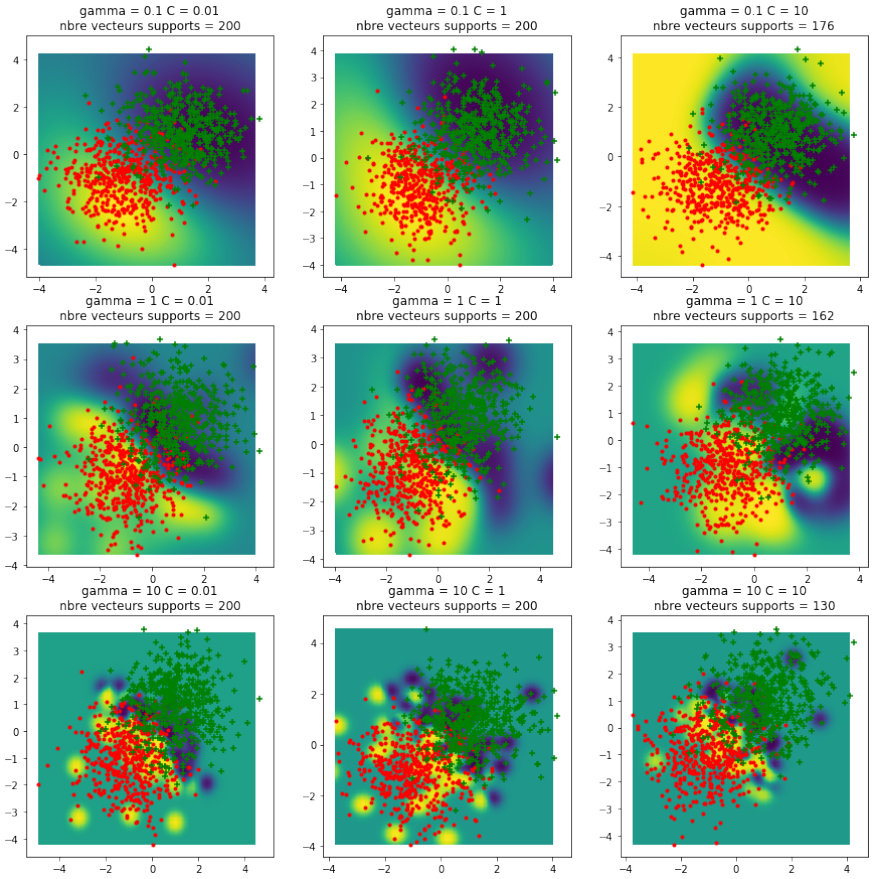


FIGURE 35 – Classification SVM sur données bruitées | noyau rbf | en variant le C et le gamma

La performance du noyau rbf dépend énormément du paramètre γ. Pour les petites valeurs de γ le modèle n'arrive pas à apprendre "assez" les données. Pour les grandes valeurs de γ on observe que presque chaque point est encerclé, c'est du sur apprentissage. Le paramètre de régularisation C n'a pas grand effet comparé au γ. Les meilleurs résultats sont obtenus avec des valeurs intermédiaires de γ et de C, respectivement 1 et 1 avec une erreur de 0.089 pour le test. Quant au nombre de vecteurs supports, on remarque que plus C croit plus ce nombre diminue ce qui est tout à fait logique, puisqu'une grande valeur de C indique qu'on favorise l'accuracy à la taille de la marge, on obtient une coupe nette qui épouse les données et une petite marge qui contient peu de points.

Résultats de tests sur données USPS :

— $C = 1$
— $d = 3$
— $\gamma = scale$

Data	Linear	Polynomial	RBF
Train	0.008504	0.005075	0.006446
Test	0.080219	0.062780	0.054310

4.2.2 Grid Search

Grid search est utilisée pour trouver les hyperparamètres optimaux d'un modèle qui donne les prédictions les plus précises. Après avoir trouvé ces hyperparamètres pour chaque modèle et pour chaque type de données. Nous avons expérimenté les modèles.

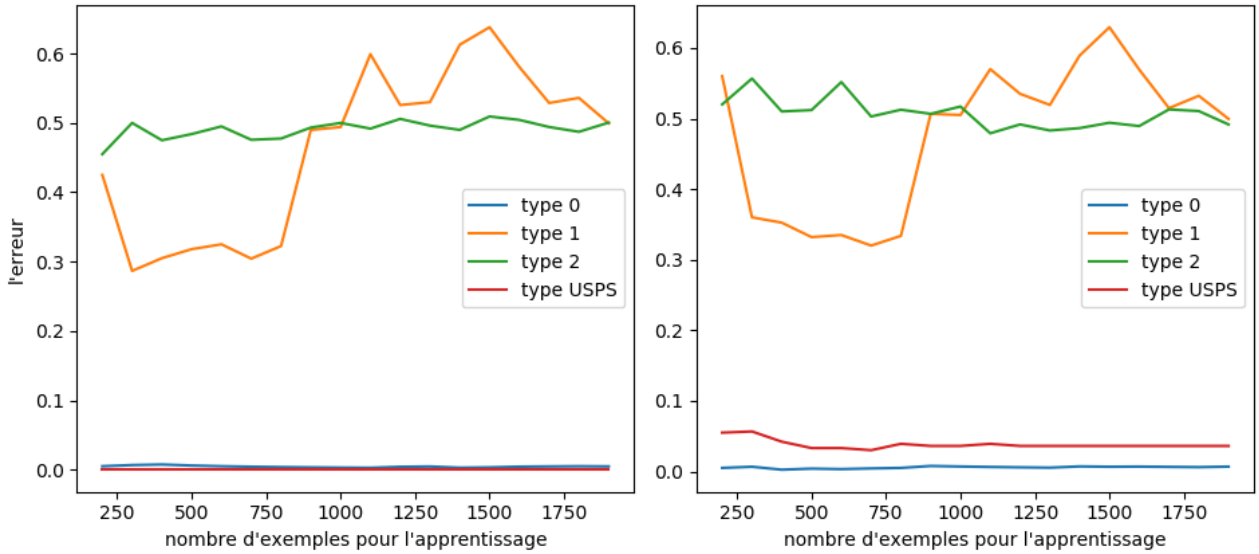


FIGURE 36 – Courbes d'erreur pour le kernel linear

le noyau linéaire fonctionne bien lorsque les données sont linéairement séparables (type de données 0). Mais une fois que les données ne sont pas séparables linéairement, le modèle ne converge plus, il continue à osciler, et donc l'entraînement sur un petit nombre d'exemples donne de meilleurs résultats.

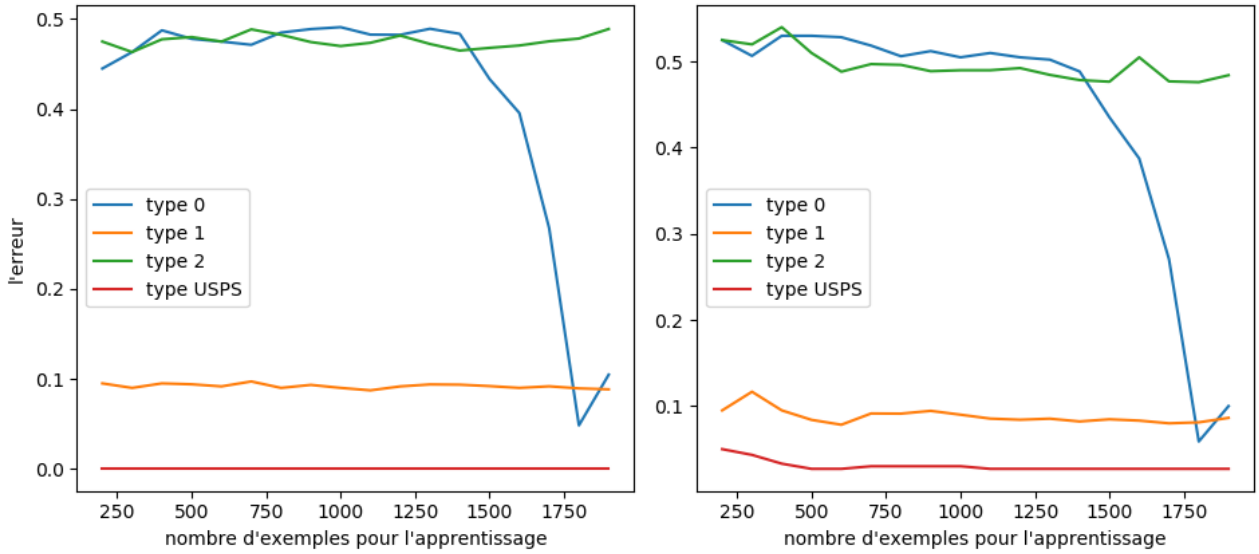


FIGURE 37 – Courbes d'erreur pour le kernel polynomial

le noyau polynomial examine non seulement les caractéristiques données des échantillons d'entrée pour déterminer leur similarité, mais aussi des combinaisons de celles-ci. Ce qui permet l'apprentissage de modèles non linéaires. Cela explique la diminution de l'erreur pour les données XOR (type 1), le noyau a augmenté la dimension afin de rendre les données linéairement séparables. Mais cela ne fonctionne plus avec les données d'échiquier (type 2) car les données ne peuvent pas être linéairement séparables quelque soit la dimension. Comme il consomme donc plus de données pour mieux apprendre.

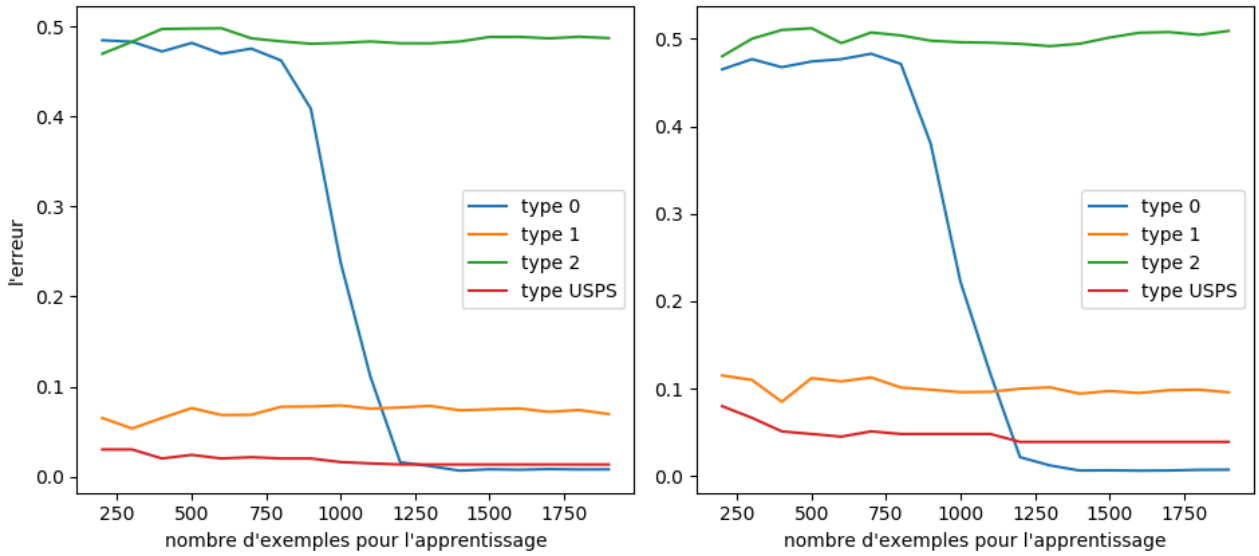


FIGURE 38 – Courbes d'erreur pour le kernel guassian

le noyau gaussien offre les meilleurs résultats pour les données linéairement séparables et même pour les données Xor (type 1) car il augmente la dimension de l'entrée. Mais les prédictions pour les données d'échiquier restent proches du hasard.

4.3 Apprentissage multi-classe

Dans cette partie, nous allons explorer le problème de la classification avec au moins 3 classes (multi-class). L'une des méthodes les plus utilisées consiste à transformer le problème en une classification binaire, car il existe déjà de nombreux classificateurs binaires avec des stratégies différentes. Parmi ces stratégies on trouve :

— **One-vs-All (OvR ou OvA)** : La stratégie consiste à former un seul classifieur par classe, les observations de cette classe étant des observations positives et tous les autres sont des observations négatives. Cette stratégie exige que les classificateurs de base produisent un score de confiance réel pour leur décision, plutôt qu'une simple étiquette de classe. À la fin, la classification se fait en considérant le classificateur ayant obtenu le score le plus élevé.

— **One-vs-One (OvO)** : La stratégie consiste à former $\frac{K(K-1)}{2}$ classifieurs binaires pour un problème multiclasse K-way. Chacun reçoit les observations d'une paire de classes de l'ensemble de l'apprentissage, et doit apprendre à distinguer ces deux classes. Au moment de la prédiction, un système de vote est appliqué : tous les $\frac{K(K-1)}{2}$ classifieurs sont appliqués à une observation non vue et la classe qui a obtenu le plus grand nombre de prédictions "+1" est prédite par le classifieur combiné.

Pour comparer les deux stratégies sur l'ensemble de données USPS, qui contient 10 classes au total, nous

avons utilisé le package **multiclass** de sklearn, avec le classifieur linéaire SVM.

$Data$	$OneVsOne$	$OneVsAll$
$Train$	0.00013	0.0111
$Test$	0.07573	0.09965

FIGURE 39 – Evaluation des stratégies OnevsOne et OnevsAll sur les données USPS

4.4 String Kernel

Dans cette partie, nous avons une base de citations avec auteurs (deux auteurs) et nous voulons utiliser l'algorithme SVM pour classifier une citation. À fin d'accroître l'expressivité de nos citations, nous allons utiliser un noyau "String Kernel" décrit dans le TME. Qu'on note pour un couple de citations $K_n(s,t)$ On construit la matrice de similarité i.e pour chaque citation s du train on calcule $K_n(s,t)$ pour toutes les autres citations t du train.

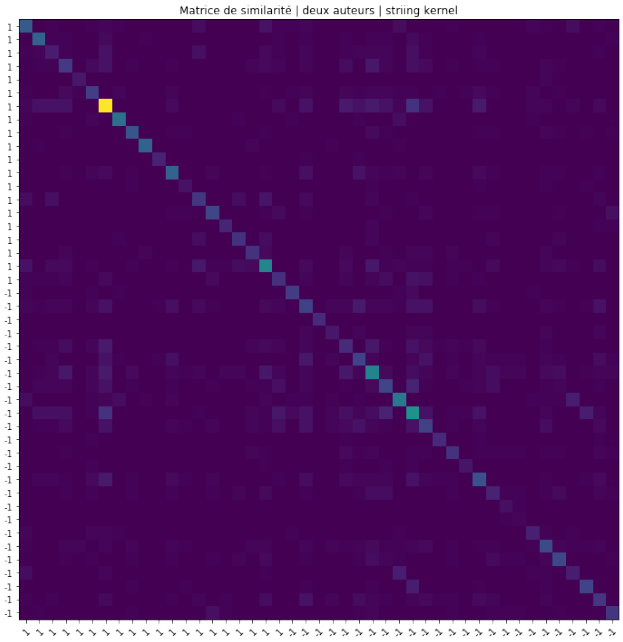


FIGURE 40 – Matrice de similarité | String kernel

Cette figure représente la matrice de similarité obtenue sur le train en utilisant le string kernel. On remarque une forte similarité sur la diagonale ce qui est triviale.

Donc au lieu d'apprendre sur le texte brut, on apprend sur la matrice de similarité ci-dessus, c'est-à-dire qu'on passe la matrice de similarité au fit du modèle au lieu des citations. On obtient une accuracy de 0.6 sur le test (sans gaps).