Sorbonne université

Rapport de ARF

Auteurs :

Marlot Maxime
Thomas Zanivan

Encadrant :

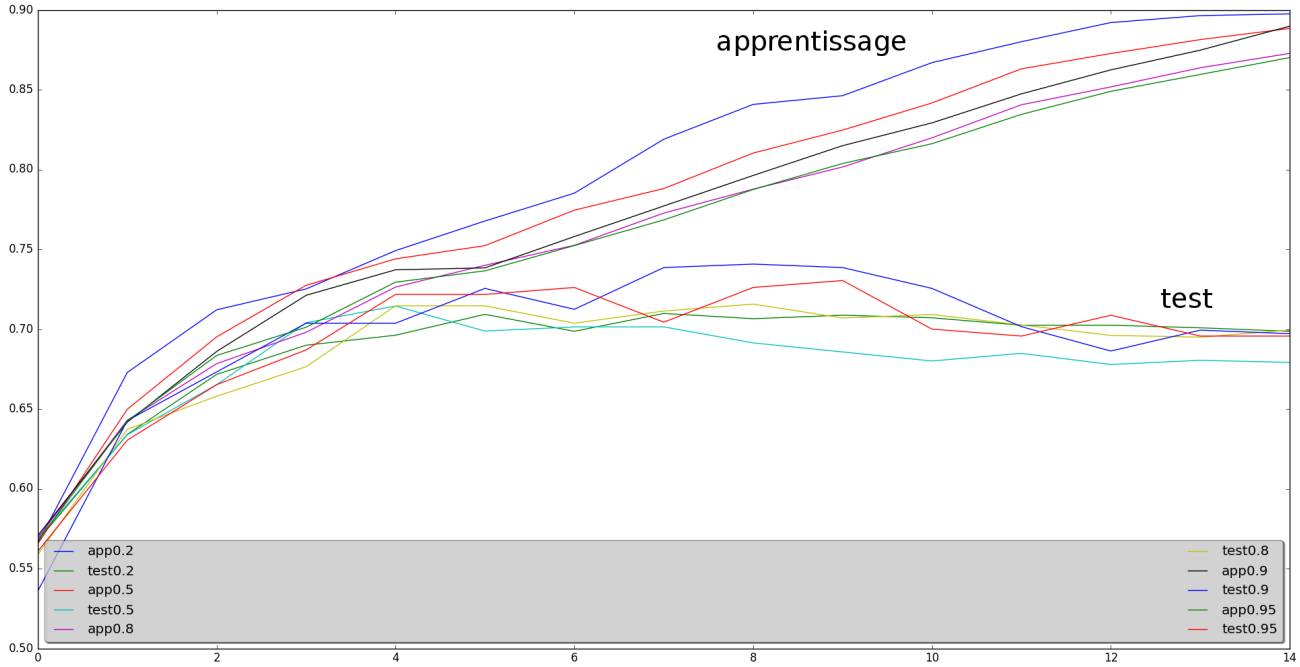
Nicolas Baskiotis

Année 2017-2018

1 Introduction

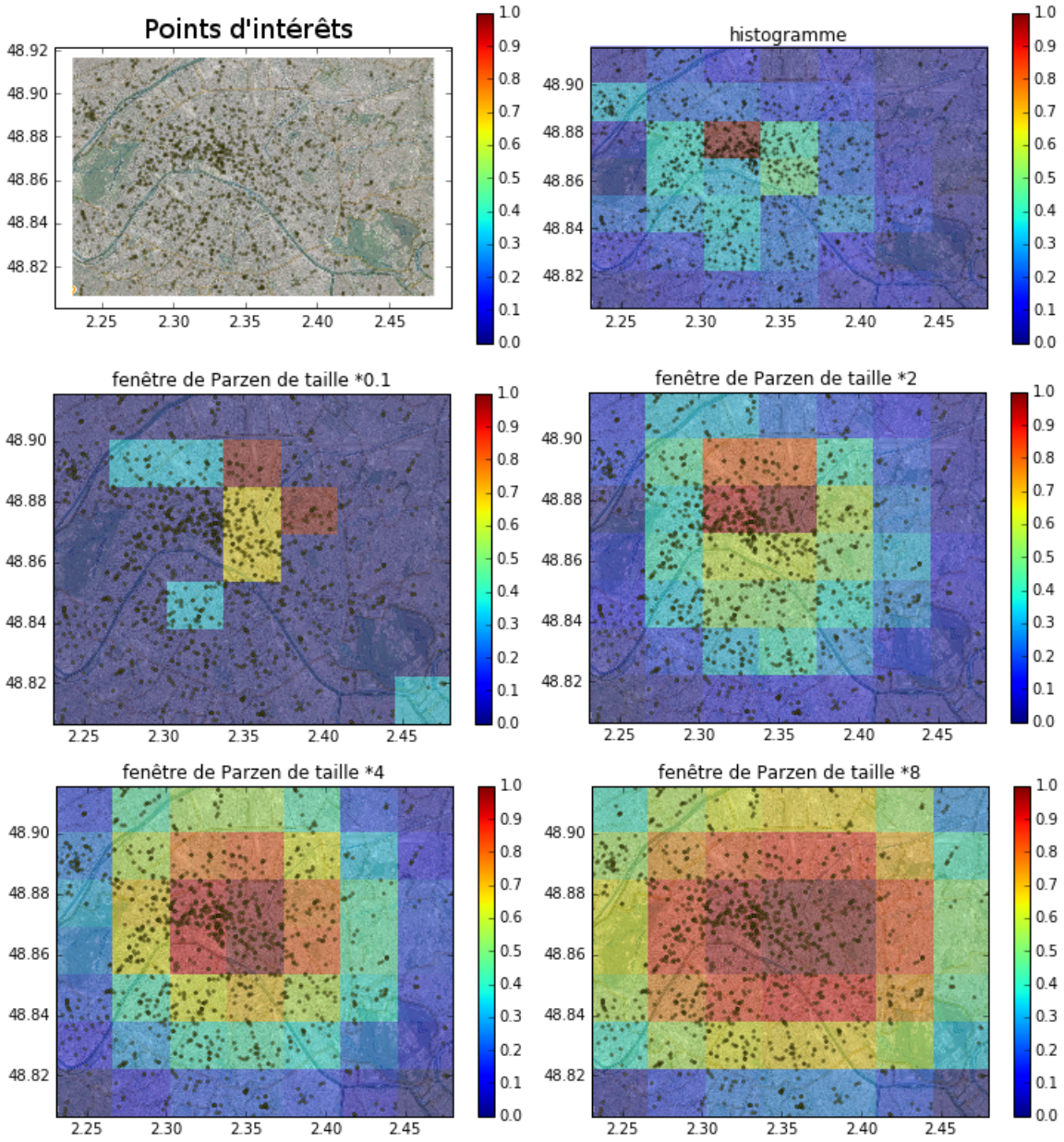
Dans ce rapport nous avons décidé de nous concentrer sur les TME sur le perceptron, ainsi que sur le SVM avec les noyaux. Voici donc nos résultats ainsi que nos interprétations.

2 tme1, Arbres de décision



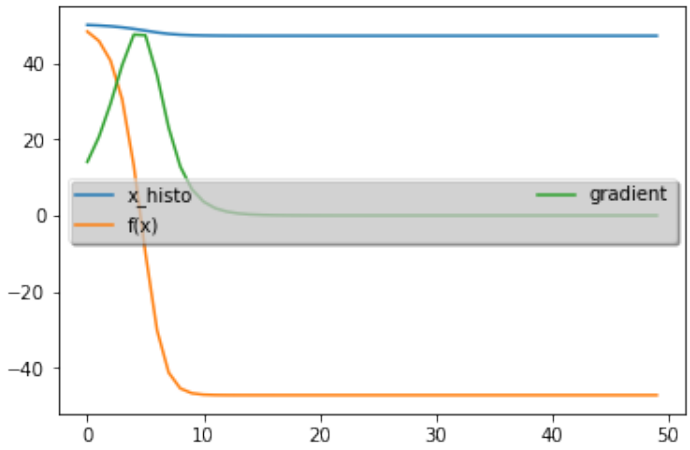
Ce graphique représente le score en ordonnée et la profondeur de l'arbre en abscisse. On a pu constater que faire augmenter la profondeur de l'arbre ne donne pas nécessairement de meilleurs scores en test après une certaine profondeur. On observe une stabilisation du score autour de 70% de réussite pour une profondeur de 4 ou 5 puis une très légère baisse si la profondeur augmente. En effet on peut observer sur le score d'apprentissage qu'on a alors tendance à sur-apprendre notre modèle. Nous avons aussi constaté que faire varier la quantité du dataset alloué à l'apprentissage et au test ne change que très légèrement le score entre 20 et 95%, ce qui est probablement du au fait que l'on dispose ici de beaucoup de données dans notre dataset.

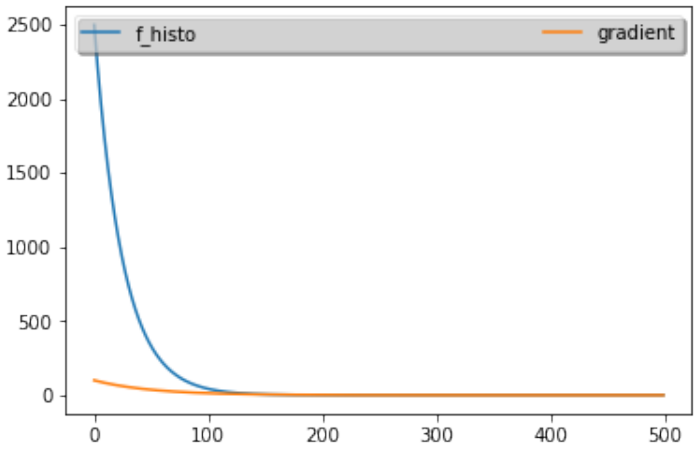
3 tme2, Estimation de densité



La fenêtre de parzen permet d'avoir une estimation de densité plus intéressante qu'un simple histogramme, notamment par le fait de pouvoir faire varier la taille de la fenêtre. Cependant on peut constater que l'estimation de densité de parzen forme des "pics" avec une taille de fenêtre trop petite (cas de sur-apprentissage) alors qu'avec une fenêtre plus grande on obtient une densité plus lissée qui généralise mieux les données.

4 tme3, Descente de gradient





On a pu constater que notre implémentation de la descente de gradient permet bien de trouver le minimum de fonctions convexes de manière itérée (cf courbes ci-dessus où le gradient converge vers 0).

5 Perceptron (tme4&5)

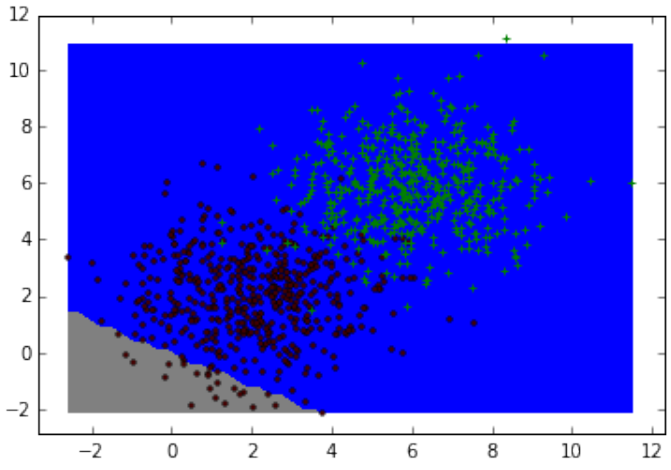


FIGURE 1 – Plot de la frontière de décision avec nbData=1000, bruit=0.7, dataType=0, SANS biais

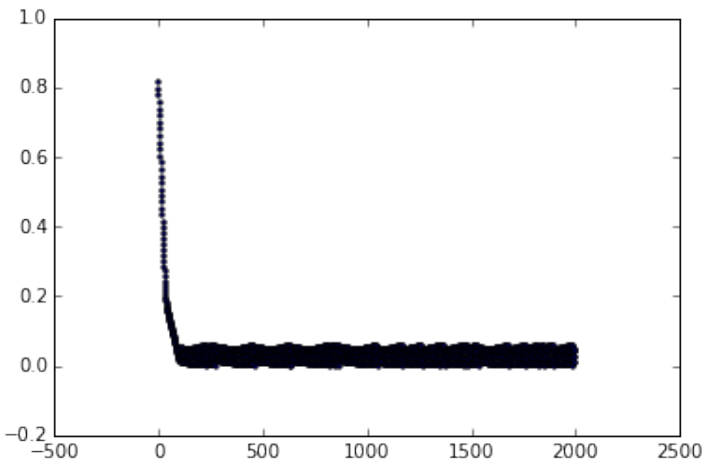


FIGURE 2 – Descente de gradient associée

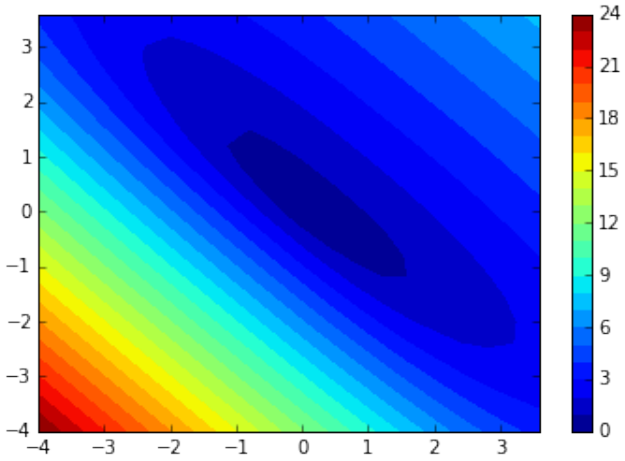


FIGURE 3 – Erreur associée

Le premier test consiste a essayer de construire un classifieur avec une frontière de décision linéaire. Pour mieux voir la différence entre le modèle avec et sans biais nous décidons de translater les données dans notre espaces afin qu'elles ne soient plus centrées en 0. Avec cette configuration nous obtenons un score de bonne classificaton sur notre data set de test de 1000 données de 0.52. On peut être étonné de ce score au vue de la frontière de décision mais étant donné que la distribution de la classe de nos données est uniforme, même en attribuant le même label à tout notre data set en prédiction, on obtiendra à minima un score de 0.5, c'est-à-dire que juste la moitié des points sont bien classés, il faut donc interpréter ce score de 0.5 comme le pire cas.

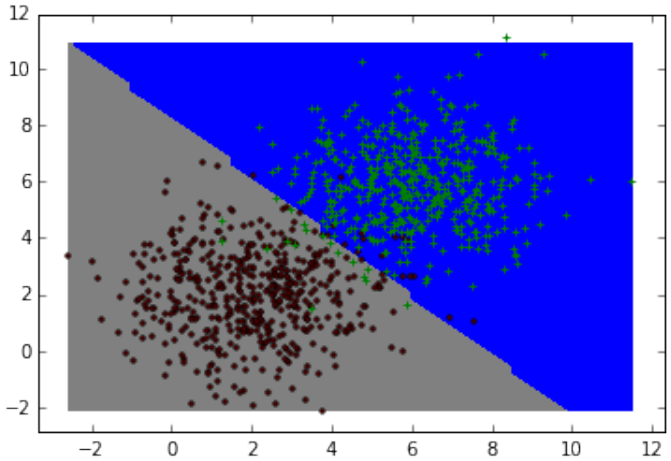


FIGURE 4 – Plot de la frontière de décision avec nbData=1000, bruit=0.7, dataType=0, AVEC biais

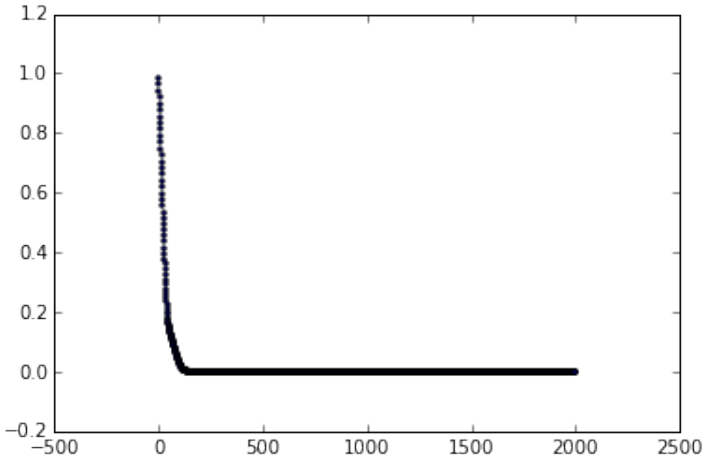


FIGURE 5 – Descente de gradient associée

Avec l'ajout du biais on obtient logiquement un meilleur score, ici les données ne sont pas tout à fait linéairement séparables du au bruit et on monte à un score de 0.96.

Toujours avec le même data set, on s'intérésse maintenant à savoir si la projection dans un espace de plus grande dimension est intéréssant dans notre cas.

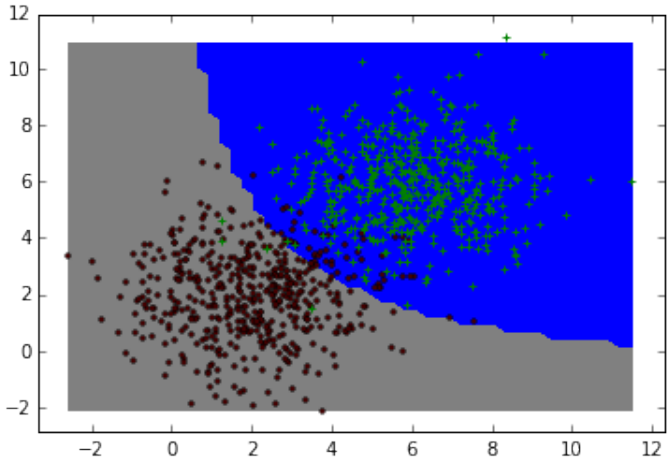


FIGURE 6 – Plot de la frontière de décision nbData=1000, bruit=0.7, dataType=0, projection=4 dimensions

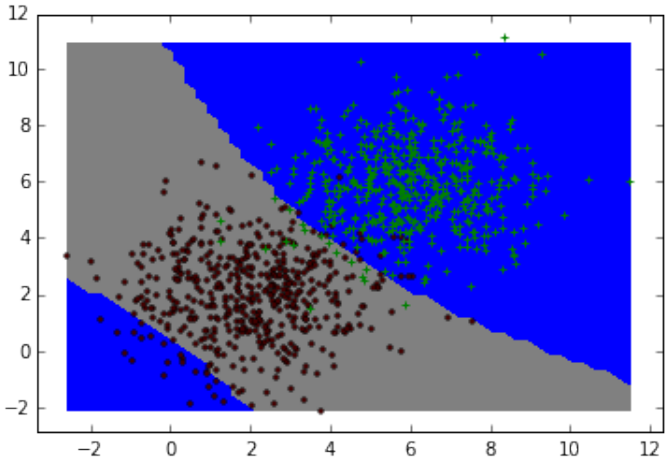


FIGURE 7 – Plot de la frontière de décision nbData=1000, bruit=0.7, dataType=0, projection=6 dimensions

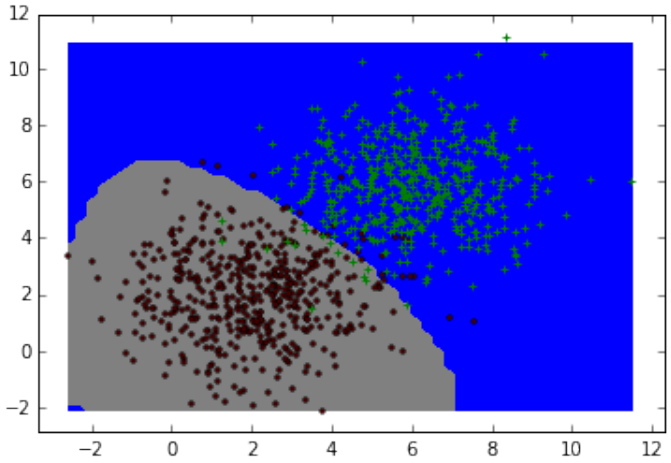


FIGURE 8 – Plot de la frontière de décision nbData=1000, bruit=0.7, dataType=0, projection=gaussien

La version de projection d'ordre 2 avec 6 dimensions ne fait pas mieux que celle avec juste 4 dimensions dont x*y qui est le seul d'ordre 2 (score de 0.94). La projection gaussienne permet d'avoir une séparation légèrement plus intéressante en revanche (0.96). Cette observation n'est pas très étonnante étant donné que comme nous pouvons le voir les données sont quasi-linéairement séparable donc l'utilisation d'une projection polynomiale n'est pas importante.

Un dernier point que nous avons voulu expérimenter est la sensibilité des projections en fonction du bruit et du temps d'exécution pour savoir quelle projection utiliser. Pour ça nous utilisons un data set qui est fabriqué à partir de 4 gaussiennes avec des labels qui empêchent la séparation de manière linéaire (contrairement à précédemment).

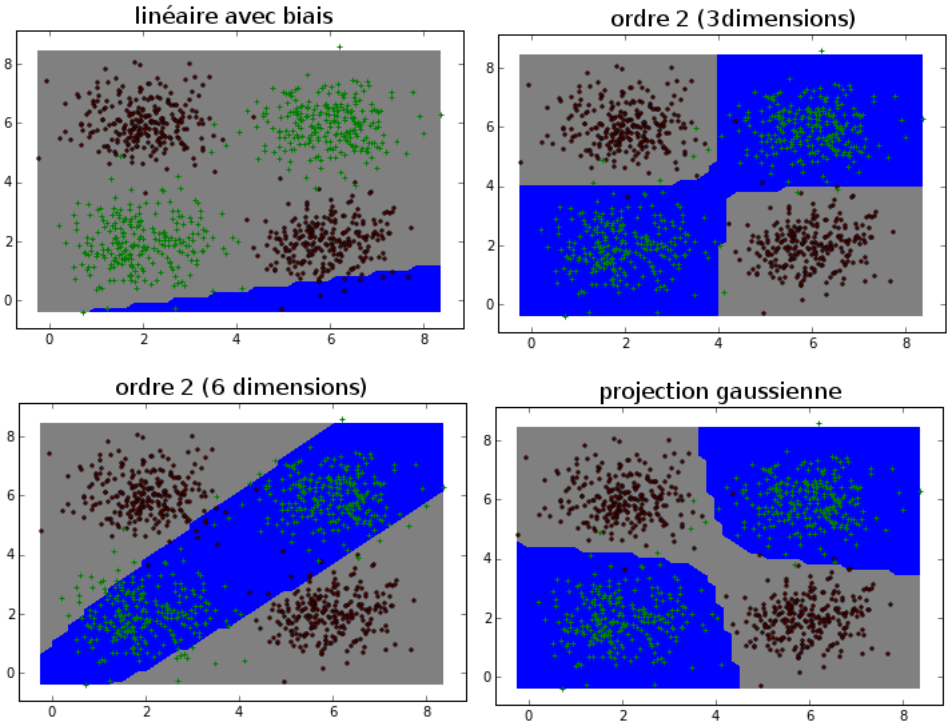


FIGURE 9 – différentes projections avec des données non-linéairement séparables

On fait alors augmenter le bruit itérativement.

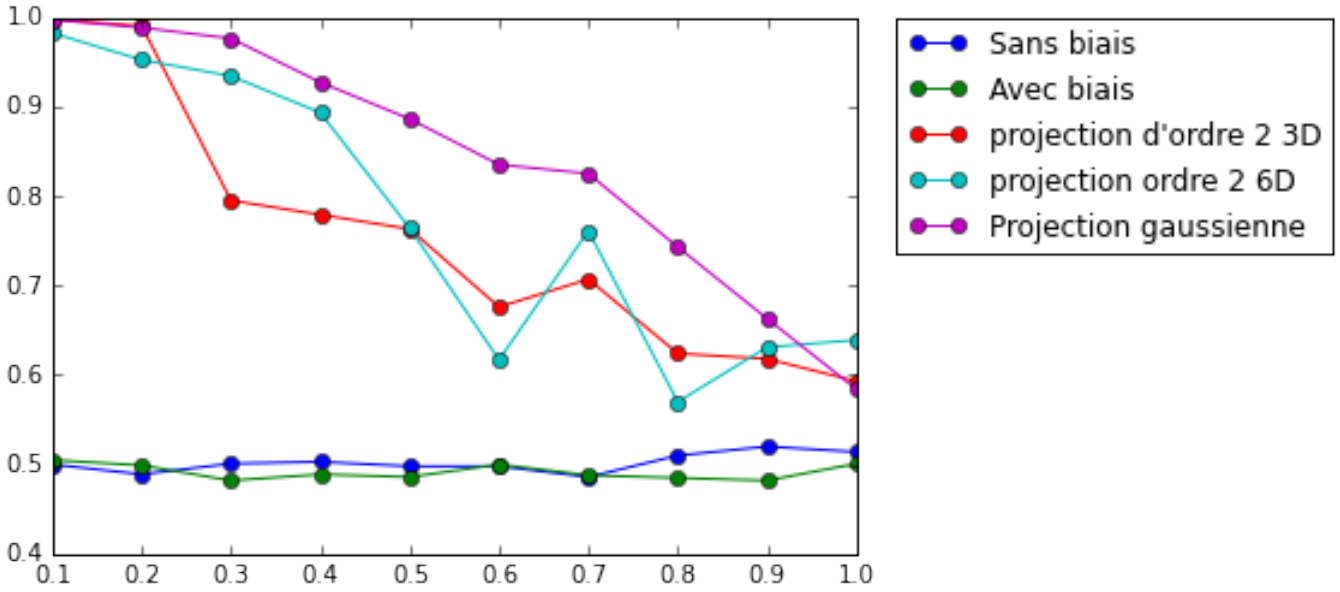


FIGURE 10 – Sensibilité au bruit avec plusieurs projections

On remarque que la projection gaussienne est celle qui semble le mieux résister à l'augmentation du bruit, aussi longtemps que c'est possible.

TABLE 1 – Temps de calcul en secondes

Projection	Sans biais	Avec biais	Ordre 2, 3D	Ordre 2, 6D	Gaussien
Temps pour l'apprentissage	0.27	0.23	0.24	0.26	26.39
Temps pour la prédiction	0.01	0.01	0.01	0.01	9.40

On remarque que le temps d'apprentissage est nettement supérieur pour le gaussien que pour les autres projections, cela s'explique par la méthode de calcul qui estime la distance de chaque point par rapport aux autres.

On peut en conclure qu'en pratique une projection d'ordre 2 donne des résultats corrects pour un temps de calcul raisonable, mais si on soupconne qu'il y ait beaucoup de bruit dans nos données il devient intéressant de considérer de prendre plus de temps de calcul pour faire une projection gaussienne.

6 SVM (tme6)

6.1 GridSearch

Pour GridSearch, nous voulons tester l'évolution du choix des paramètres, ici le paramètre C, ainsi que le choix du kernel (linéaire, gaussien, polynomial de degré 3 ou sigmoid) en fonction du bruit avec 1000 données distribuées sous forme de deux gaussiennes, nous pensions que le grid search choisirait le kernel linéaire au départ puis RBF (voir uniquement RBF dès le départ) mais nous nous rendons compte qu'avec beaucoup de bruit il "se perd" et commence à revenir au linéaire, après réflexion, cela semble logique puisqu'avec beaucoup de bruit, le score sera semblable peu importe le kernel. On remarque également que C augmente fortement à partir d'un bruit de 2.5, ce qui signifie que le SVM va chercher à pénaliser les points mal classés car plus C est grand et moins le modèle sera tolérant aux fautes (ce qui signifie aussi qu'il y a un risque de sur-apprentissage) à l'inverse plus C sera petit, plus on sera tolérant et par conséquent on limitera le sur-apprentissage.

TABLE 2 – Experience pour les paramètres du grid search

Bruit	C	Kernel	Score
0.0	0.01	Linéaire	1
0.5	0.01	Gaussien	0.97
1.0	0.01	Gaussien	0.90
1.5	0.01	Sigmoide	0.82
2.0	0.01	Linéaire	0.80
2.5	1.41	Linéaire	0.72
3.0	3.11	Linéaire	0.67
3.5	3.61	Linéaire	0.65
4.0	4.1	Linéaire	0.63

6.2 Analyse du score en fonction du type des données et du bruit

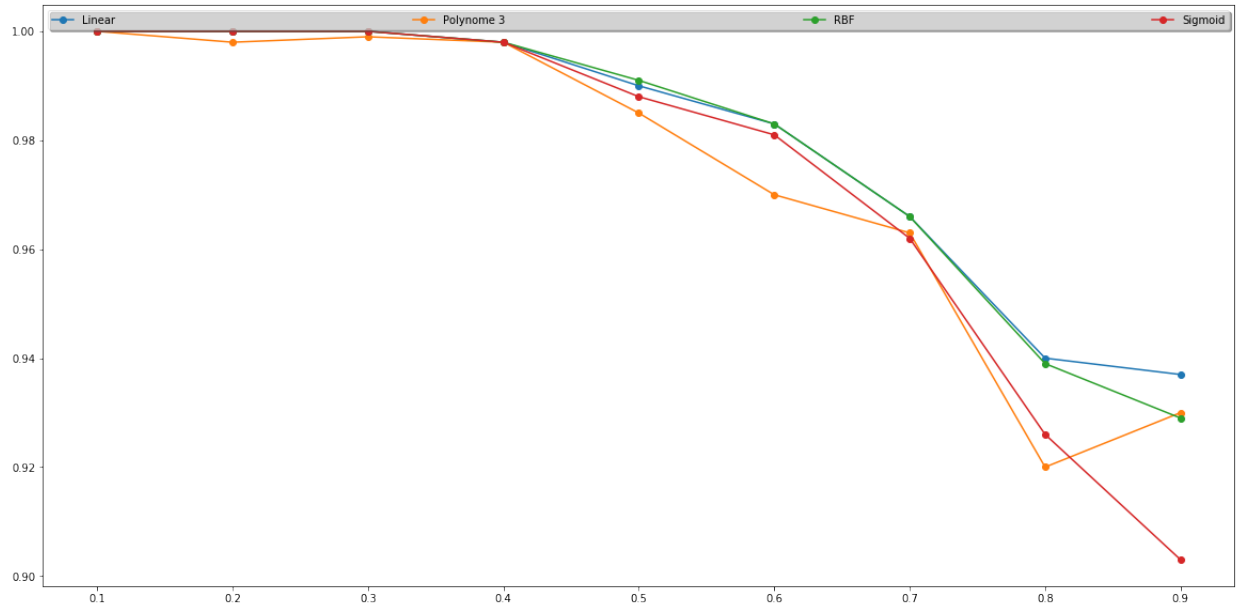


FIGURE 11 – score du SVM sur des données linéairement séparables en fonction du bruit

Sur des données linéairements séparables les différents noyaux obtiennent tous de bon scores, mais le noyau sigmoide semble avoir plus de mal à tolérer le bruit, alors que les noyaux linéaire et RBF le tolère mieux.

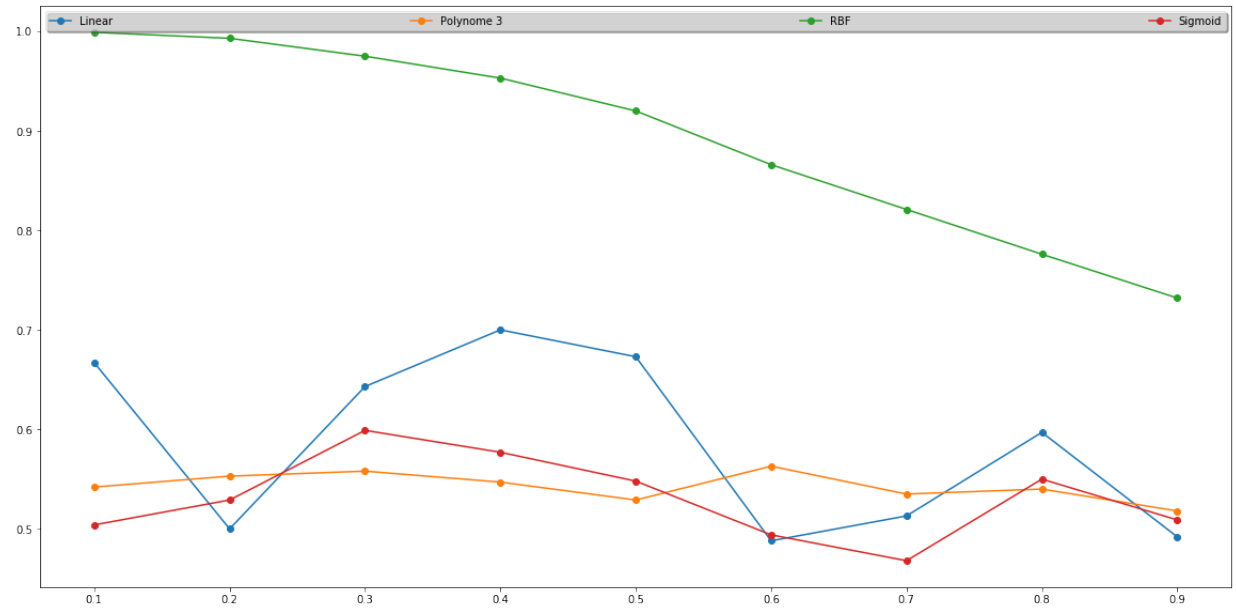


FIGURE 12 – score du SVM sur des données non-linéairement séparables en fonction du bruit

Sur des données non-linéairement séparables on constate que le noyau RBF est le seul à obtenir un bon score et à avoir une bonne résistance au bruit.

On peut en conclure que le noyeau RBF est celui à utiliser en priorité pour sa versatilité et sa bonne tolérance au bruit.

6.3 Analyse du nombre de vecteurs supports en fonction du bruit

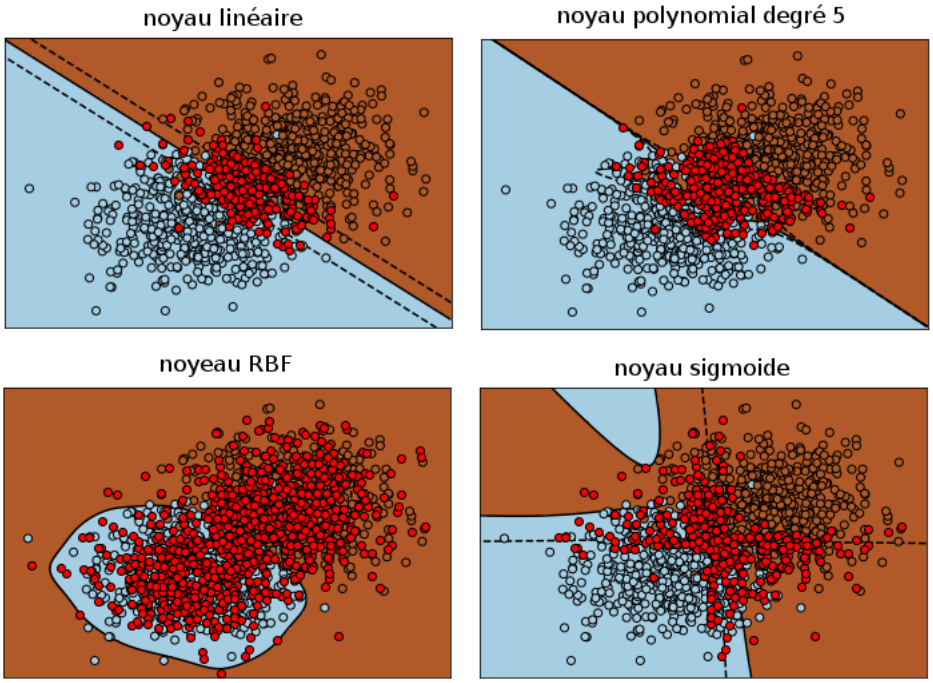


FIGURE 13 – nombre de vecteurs supports en rouge avec différentes projections sur des données linéairement séparables

Pour rappel plus C est grand et moins le modèle sera tolérant aux fautes (ce qui signifie aussi qu'il y a un risque de sur-apprentissage) à l'inverse plus C sera petit, plus on sera tolérant et par conséquent on limitera le risque de sur-apprentissage.
On remarque que quand C est petit, le SVM a tendance à utiliser un nombre important de vecteurs supports et même quasiment tous les vecteurs dans le cas du noyau RBF. Cela semble logique car si on souhaite une frontière qui n'est pas du tout tolérante aux fautes, on va uniquement chercher à choisir les vecteurs les plus proches de notre marge de manière à ce qu'il y ait le moins de points mal classés en apprentissage et on ignorera les données plus éloignées (quitte à favoriser le sur-apprentissage comme dit plus haut). Même si l'utilisation d'un C très petit avec un noyau gaussien a souvent donné des résultats satisfaisants dans nos expérimentations, il est certain que dans le cas où nous aurions des centaines de milliers de données, le processus peut devenir très couteux.

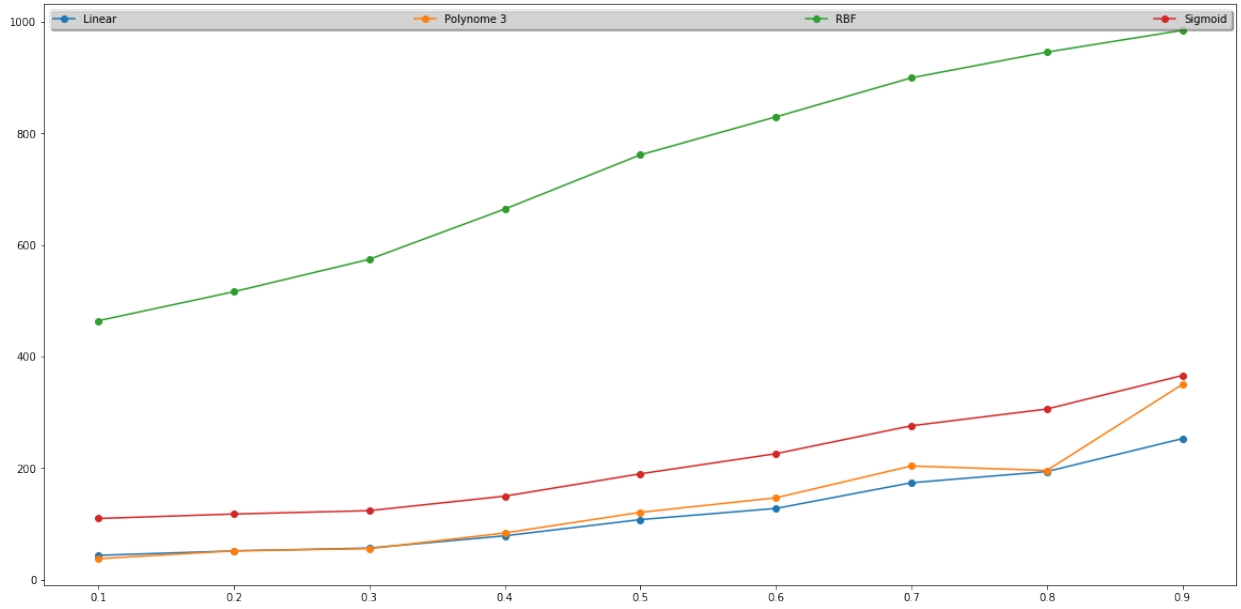


FIGURE 14 – Evolution du nombre de vecteurs support en fonction du bruit avec C petit

On observe que le nombre de vecteurs supports augmente avec le bruit de façon linéaire et assez lente quand C est petit, le noyau RBF en utilisant beaucoup plus en moyenne.

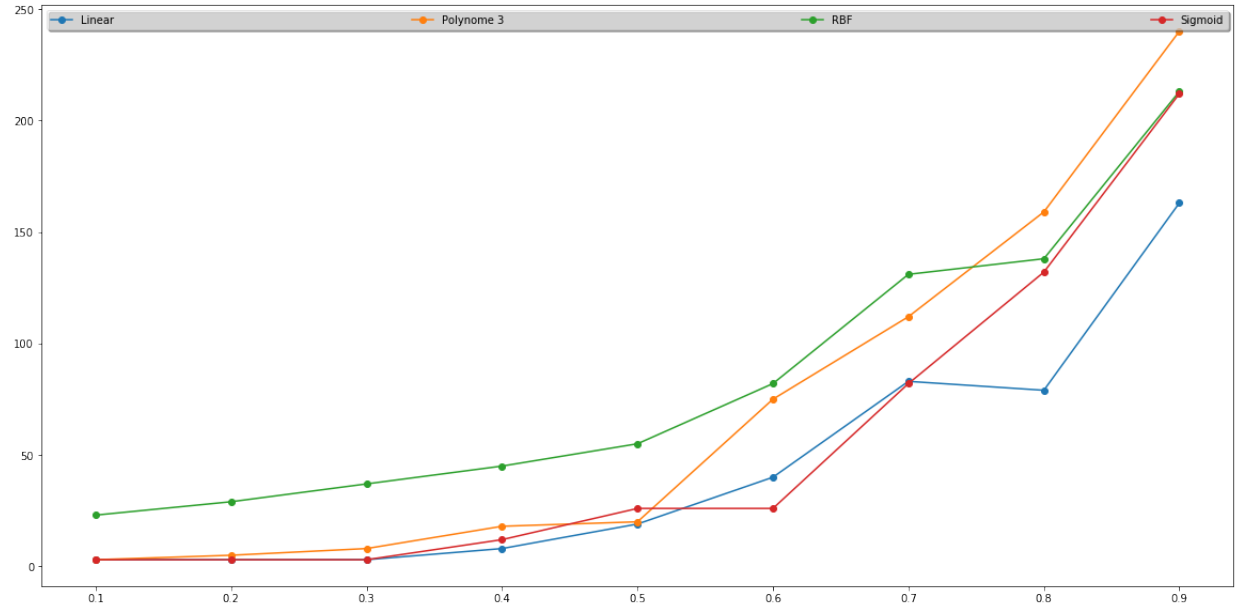


FIGURE 15 – Evolution du nombre de vecteurs support en fonction du bruit avec C élevé

Avec un C grand, le noyau RBF en utilise toujours plus lorsque le bruit est faible, mais cette différence tend à s'estomper quand le bruit augmente, sauf pour le noyau linéaire qui en utilise toujours moins. Par contre le nombre de vecteurs-supports augmente de façon exponentielle avec le bruit quand C est grand pour tous les types de noyaux.