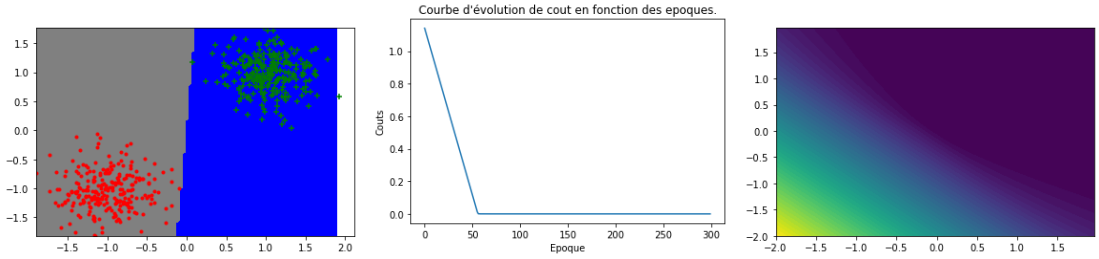
TME 4 - Perceptron

CHERCHOUR Liècе & DIEZ Marie

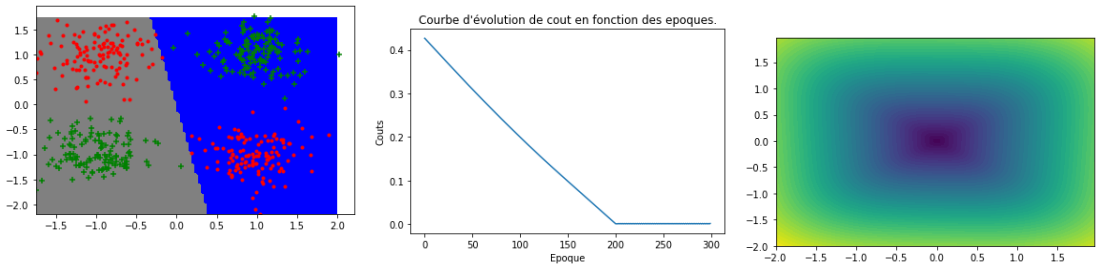
Avec l'aide de THAUVIN Dao & STERKERS Luc

1 Algorithme du perceptron sur les données de gen arti

1.0.1 Tests sur les données de type 0



1.0.2 Tests sur les données de type 1

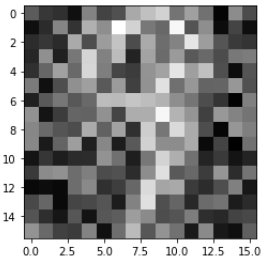


1.1 Observations

On remarque que le perceptron détecte bien les 2 nuages de points sur les données de type 0, la droite séparatrice est proche des nuages de points car en effet le perceptron s'arrête dès qu'il a bien tout classifier sans avoir de contrainte de pénalisation sur la distance des points à la frontières (la marge). Cependant sur les données de type 1 le perceptron n'est pas capable de séparer correctement les données, celles-ci ne sont pas linéairement séparable et le perceptron est un séparateur linéaire, sans projection il est incapable de classifier de tel données.

2 Algorithme du perceptron sur les données USPS

2.1 Visualisation de la matrice des poids dans un exemple 6 vs 9



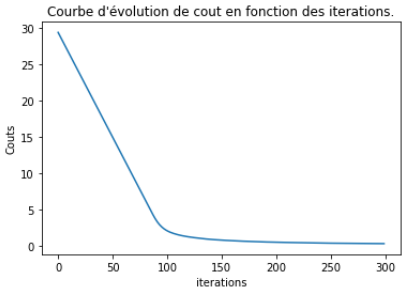
2.1.1 Observations

Sur cette matrice des poids on peut distinguer un 9, en effet nous avons dis au modèle de classifier positivement les pixels appartenant au 9 et négativement ceux appartenant au 6, les poids appris sur un ensemble d'apprentissage discrimineront donc positivement le 9 et négativement le 6, l'apprentissage des poids permet d'ajuster les réponses du modèle et mène à cette image.

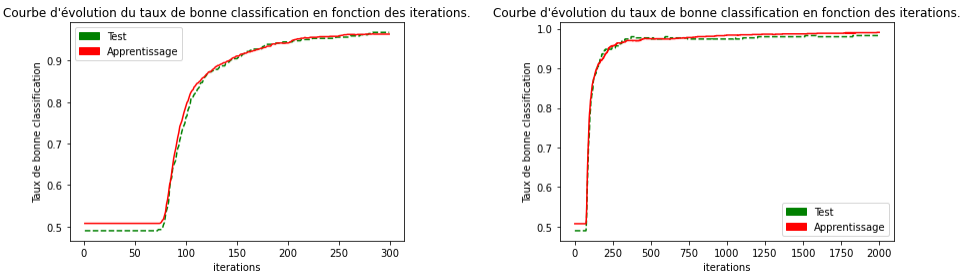
Dû à l'initialisation aléatoire des poids et le manque des pénalisations, les contours du 9 sont difficilement visibles (L'algorithme s'arrête des qu'il peut bien tous classifier, même si il pourrait trouver une frontière un peu plus idéal.

2.1.2 Analyse des courbes

2.1.2.1 Coût



2.1.2.2 Courbe d'erreur en test et en apprentissage



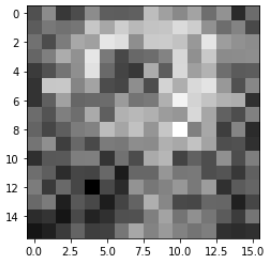
2.1.3 Observations

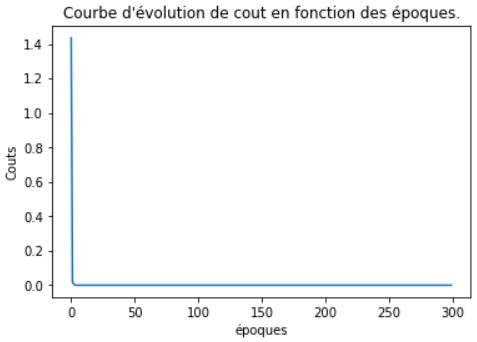
On remarque que le perceptron converge en environ 200 itérations et obtient de très bonne performance en apprentissage certes, mais aussi en test. Il n'y a pas de sur apprentissage lorsque l'on augmente le nombre d'itérations, le modèle à trouver en 200 itérations le vecteur w^* optimal et peut importe combien d'itérations on ajoute il ne modifiera pas w^*.

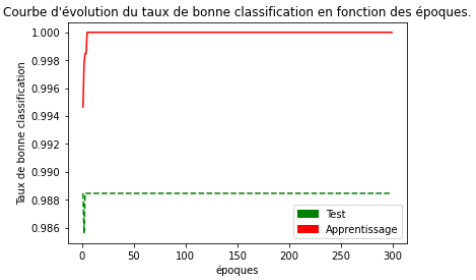
3 Mini-batch et Descente stochastique

3.1 Descente de gradient stochastique

3.1.1 Données USPS dans un exemple 6 vs 9







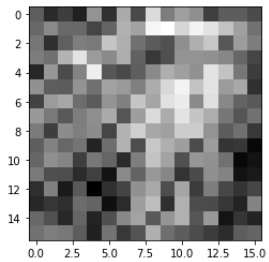
On remarque que l'image obtenu est relativement similaire à l'image obtenu précédemment, la différence ici est la rapidité de la convergence vers $w*$. En effet on remarque que l'algorithme converge très vite, cela peut s'expliquer car la méthode stochastique met à jour à chaque nouvelle donnée le vecteur de poids contrairement au batch qui le met à jour en fonction de toutes les données, si nos jeux de données sont propres et avec peu de bruit, la méthode stochastique peut converger beaucoup plus vite que la méthode précédente, lorsque que les données sont moins "facile" la méthode stochastique sera plus lente mais beaucoup plus résistante aux bruits. Une solution intermédiaire est la descente de gradient par mini-batch.

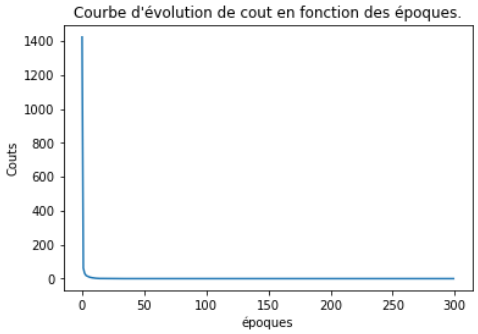
3.2 Descente de gradient par Mini-Batch

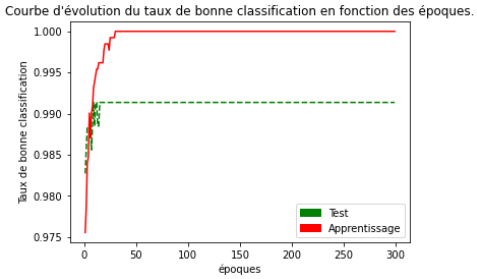
3.2.1 Courbe sur les données USPS dans un exemple 6 vs 9

On peut remarquer que la descente stochastique correspond à une descente mini-batch de taille 1, et une descente batch à une descente mini-batch de taille le nombre d'exemples. Le choix de la taille du batch $batch\_size$ est donc un choix à prendre en fonction des données et qui permet d'avoir un résultat intermédiaire entre le batch et le stochastique, voir ci-dessous.

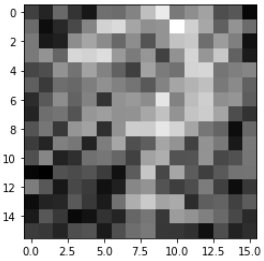
- $batch\_size = 5$

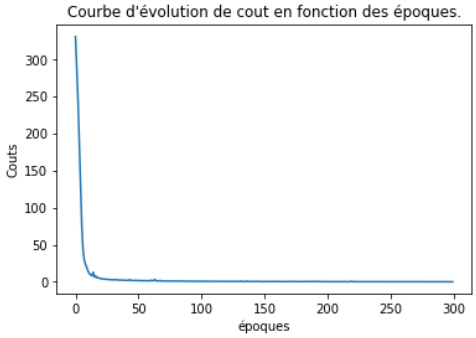


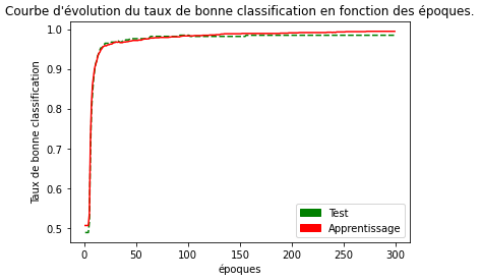




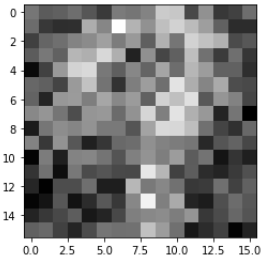
- $batch\_size = 100$

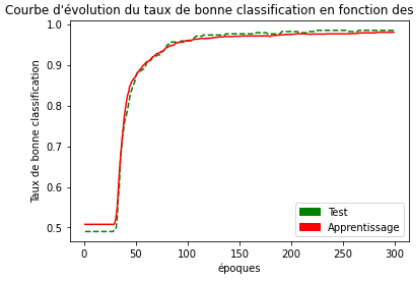


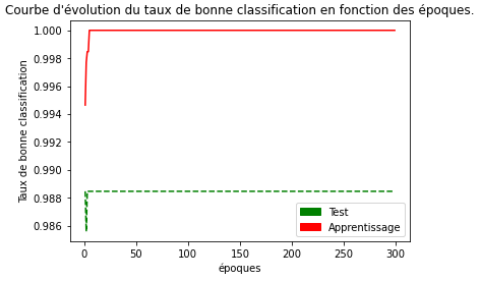




- $batch\_size = 500$



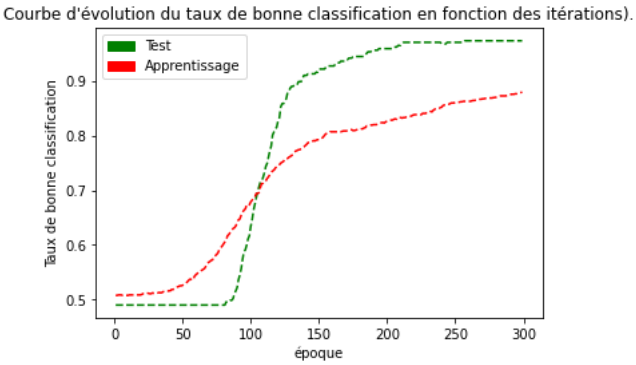




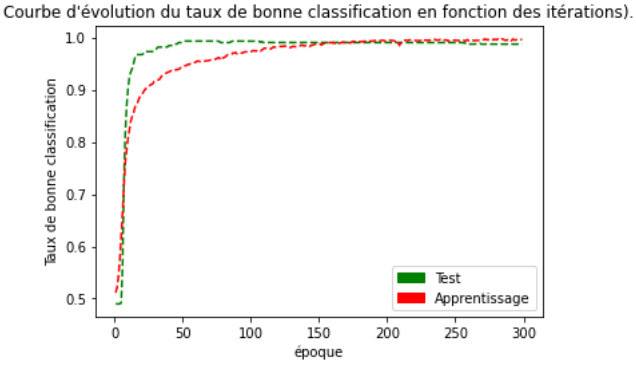
3.3 Comparaison sur des données de générations artificielles et résistance aux bruits

Pour un bruit avec $eps = 0.7$ sur gen_arti on obtient :

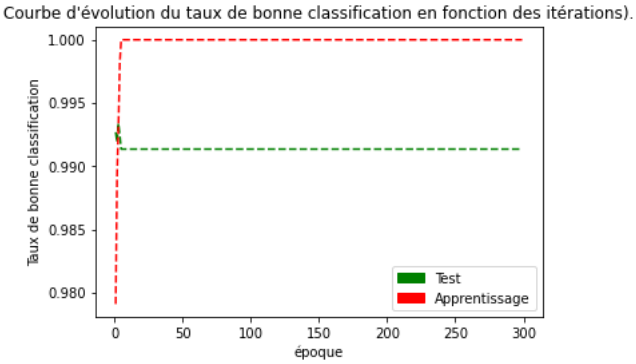
- Batch



- Mini batch de taille 100



- Stochastique



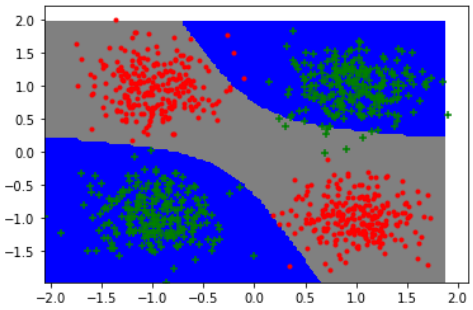
On remarque que la descente de gradient avec la méthode batch est très sensible au bruit, il y a des pics de perte qui descende jusqu'a 20% de bonne classification et le modèle n'est pas stable. Plus la taille des batchs diminue (jusqu'à $batch\_size = 1 \leftrightarrow$ Stochastique) plus le modèle est stable, l'échelle est beaucoup plus faible.

4 Projections et pénalisation

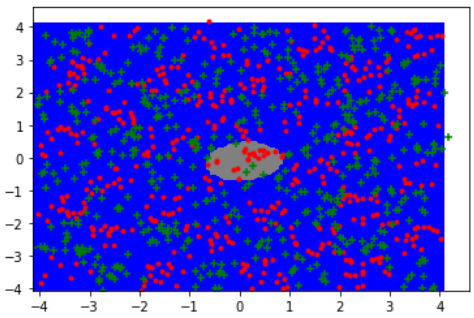
Les projections permettent au perceptron de classifier des données non linéairement séparable.

4.1 Projection polynomiale

4.1.1 Tests sur les données de type 1



4.1.2 Tests sur les données de type 2



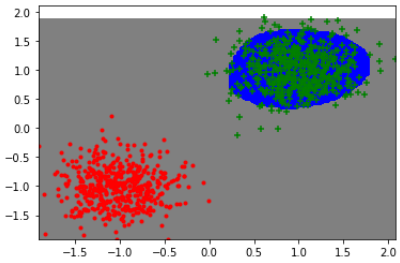
4.1.3 Observations

On remarque que la projection polynomiale marche bien sur les données de type 1 mais pas sur les données de type 2. En effet les données de type 1 sont séparables à l'aide d'une projection avec un noyau polynomiale, contrairement au problème de l'échiquier.

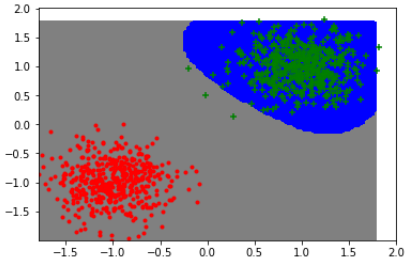
4.2 Projection Gaussienne

4.2.1 Tests sur les données de type 0

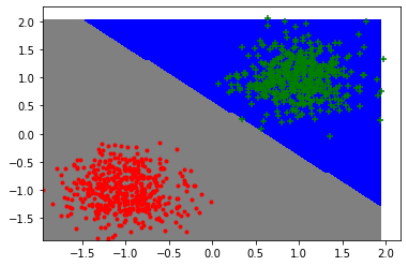
Test sur σ avec 1000 points dans la base
Notre hyperparamètre σ permet de modifier la distance de similarité, plus il est grand plus la distance de similarité est grande et inversement. Autrement dit on considère plus de points pour la décision de la frontière.
Avec $\sigma = 0.1$



Avec $\sigma = 0.4$



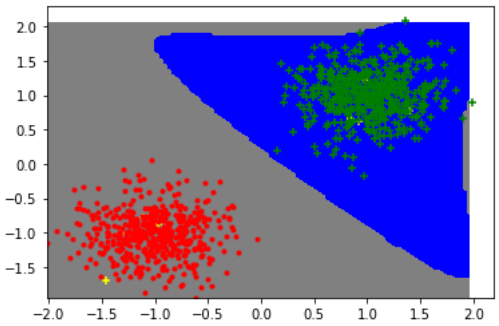
Avec $\sigma = 5$



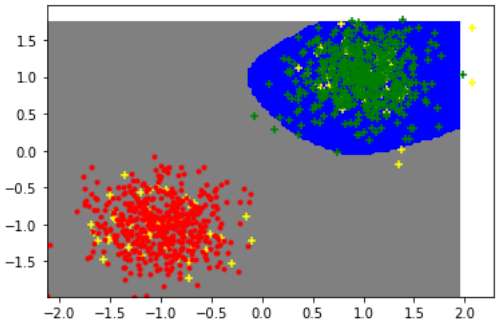
On remarque bien que l'hyperparamètre σ permet de prendre en considération plus ou moins de points pour la décision de la frontière.

Test sur le nombre de points dans la base avec $\sigma = 0.4$
Le nombre de points dans la base permet de définir combien de points "ancres" vont être choisis dans la projection, cela évite d'avoir un trop grande nombre de points à prendre en compte, plus les points ancres sont nombreux plus le nombre de points pour la décision de la frontière est important. Cependant si on prends trop peu de points la frontière risque de ne pas être assez précise.

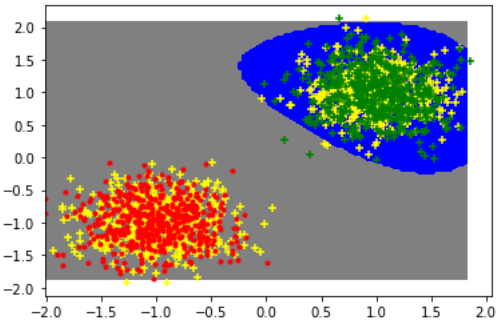
- 10



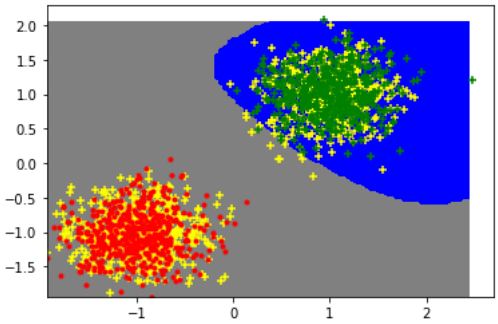
- 100



- 1000

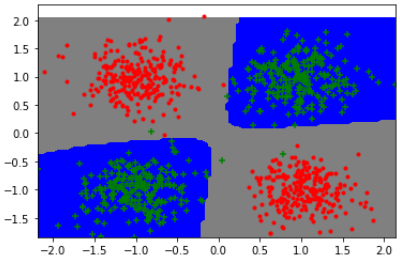


- 10000



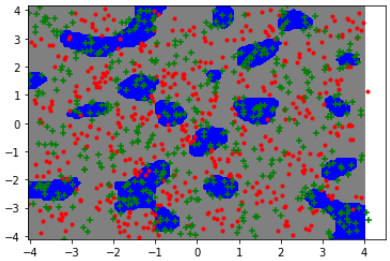
On remarque que le 1000 points dans la base est raisonnable au dessous les résultats manques de précision.

4.2.2 Tests sur les données de type 1 avec $\sigma = 0.4$ et 1000 points dans la base

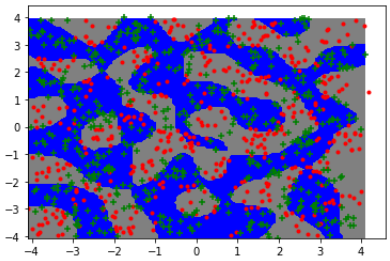


4.2.3 Tests sur les données de type 2 avec 1000 points dans la base

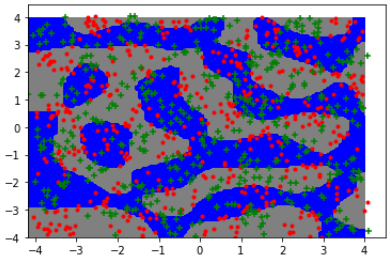
Avec $\sigma = 0.1$



Avec $\sigma = 0.4$



Avec $\sigma = 0.7$



4.2.4 Observations

On peut voir l'importance de l'hyperparmètre σ qui permet de définir la distance de similarité, il faut faire attention de ne pas choisir σ trop petit pour ne pas faire de sur-apprentissage ni trop grand pour éviter le sous-apprentissage.

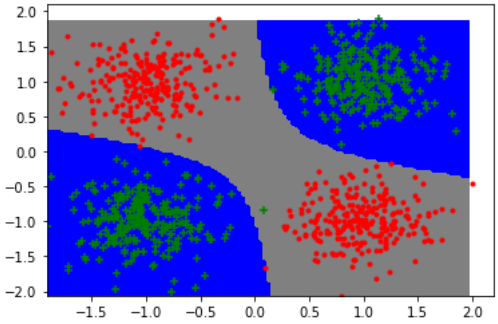
De plus, Il peut être possible dans certains cas, on peut imaginer projeter nos points dans une grille gaussienne (Par exemple ici 8 points espacés équitablement en x et 8 points en y). Cela aurait pu être une façon de résoudre le problème de l'échiquier avec un minimum d'erreur. (Mais un gros coût computationnelle dû à la projection)

4.3 Ajout d'une marge et pénalisation par la norme

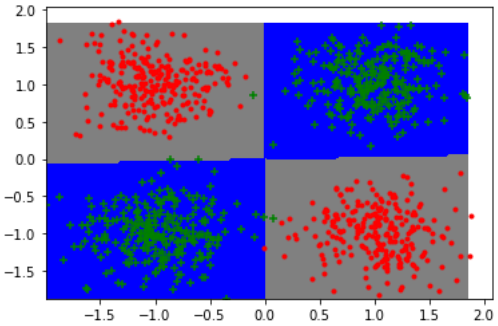
On peut également étudier l'effet d'une marge et d'une pénalisation sur les poids de la projection gaussienne avec *hinge_loss* et *hinge_grad*.

4.3.1 alpha

- $\alpha = 10^{-5}$



- $\alpha = 1$

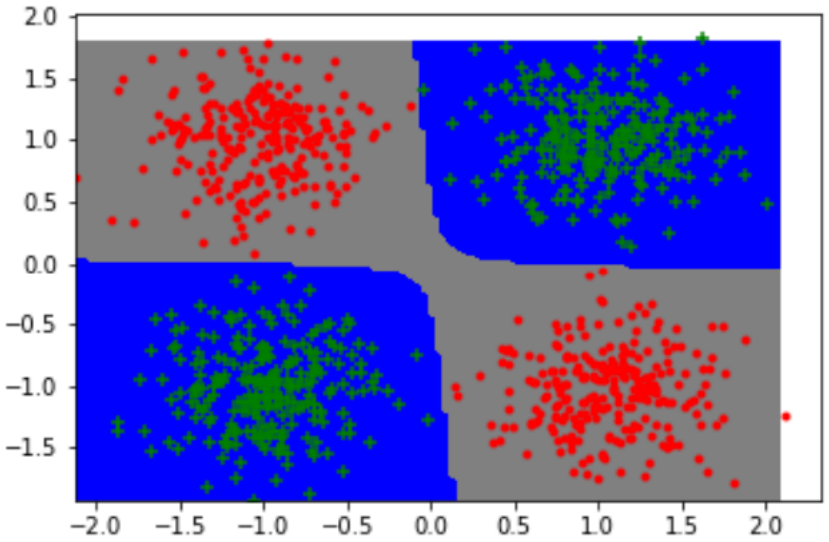


α permet de régler la distance du point le plus proche de la frontière à la frontière = la marge, plus il est grand plus les points doivent être loin de la frontière comme on peut le voir ci-dessus.

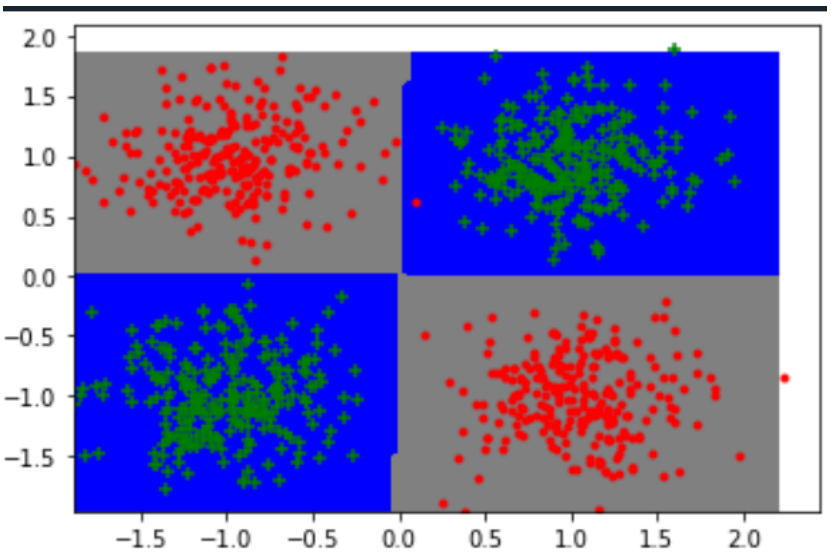
4.3.2 lambda

λ est le coefficient de pénalisation de la fonction de coût il permet de limiter l'expressivité du modèle et donc le sur-apprentissage. c'est un hyperparamètre à régler en fonction des données, il faut faire des expériences. On a réalisé des tests avec $\alpha = 1$

- $\lambda = 0.1$



- $\lambda = 10$



On remarque que cet hyperparamètre permet de limiter l'expressivité du modèle, en effet plus λ est grand, plus la contrainte est forte est donc plus le modèle est généraliste, comme on peux le voir ci-dessus.