TP Support Vector Machine (SVM)

C de la Chapelle

Introduction

Ce TP est une application du cours du 24 Septembre dans le lequel il est présenté la séparation ou classification de données binaires par la méthode d'apprentissage SVM ou Machine à vecteurs supports.

Les données observées sont des couples de variables (xi1, yi) et sont séparables par un hyperplan sous la contrainte de maximiser les marges entre l'hyperplan est les données les plus proches.

Selon la natures des données, la séparation entre les données est par exemple, si on se limite à 3 variables descriptives, plane ou courbe comme sphérique ou parabolique. Pour ce ramener à un problème linéaire, on utilise des fonctions noyau permettant de lineariser au mieux les données.

Dans ce TP, la bibliothèque python sklearn permettant de régler le paramètre C qui contrôle la complexité du classifieur car C détermine le coût d'une mauvaise classification : plus C est grand, plus la règle obtenue est complexe (le nombre de points pour lesquels on veut minimiser l'erreur de classification croît). Cette approche est appelée C-classification et s'utilise avec l'objet sklearn.svm.SVC dans le module scikit-learn.

Deux scriptes python sont associés à ce TP, svm_script.py, qu'il à fallu compléter et svm_source.py qui contient des fonctions utiles au premier script.

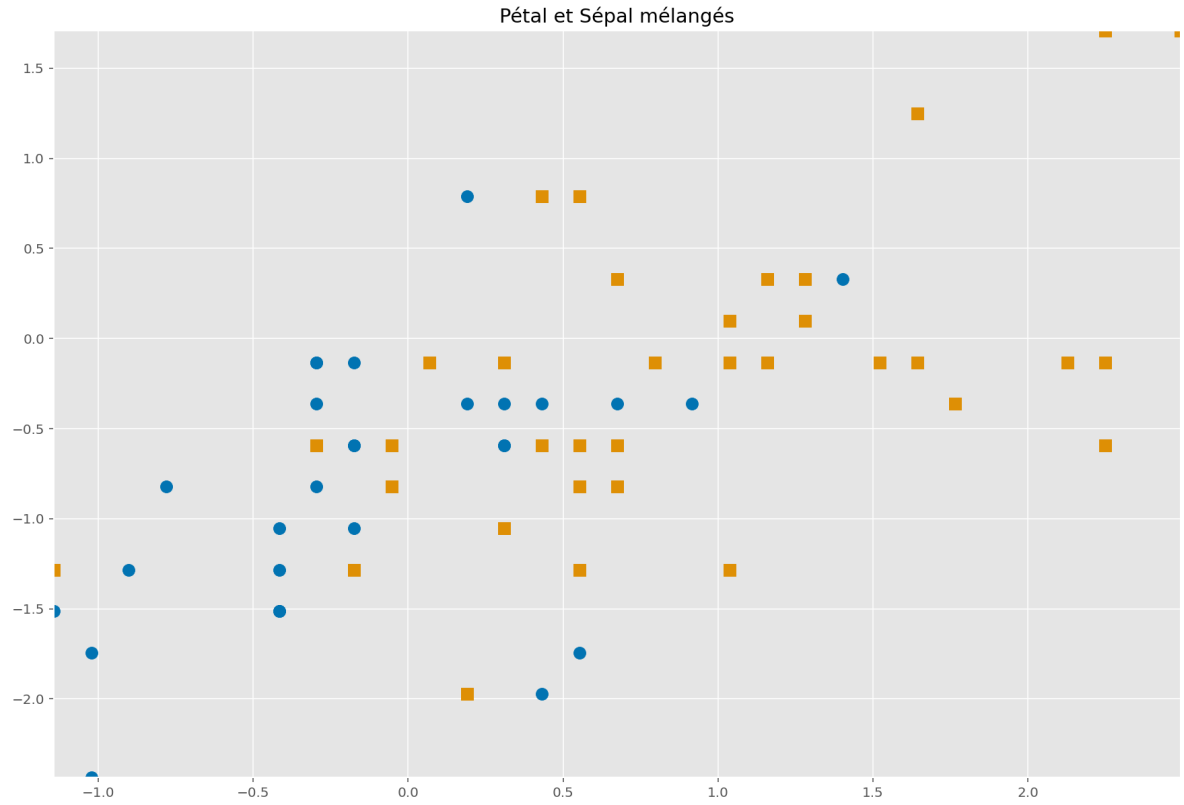
Question 1 :

Le TP commence par un exemple de 2 distributions gaussiennes un peu différentes. Très utile pour présenter le formalisme des les fonctions de Sklearn pour un débutant en Python. Connaissant le C++, le Pascal et ayant déjà vu du code, pas trop de difficultés dans ce TP mais aucune finesse dans python.

On importe Iris facilement car il est contenu dans les Packages chargés en début. Il s'agit des dimensions morphologiques des fleurs d'iris de 3 variétés sous formes 50 individus en lignes par variétés (soit 150 lignes) et 4 variables morphologies en colonnes (détails en commentaire)

On travaille sur les 2 premières variables (en colonnes, “sepal length (cm)’, ’sepal width (cm)”) et les pour les variétés (y), on exclue la premières variété. Les longueurs des variables X sont centrées réduites, les y variables nominales convertis en quantitatives centrée (+/- 1). Enfin, les échantillons train et test sont réalisés en ajoutant de l’aléatoire pour une meilleur indépendance des individus :

Dn étant ici (X-train, y_train). On procède alors à la modélisation sur l’échantillon train avec un noyau linéaire. Mais avant regardons l’allure des données :



Il est évident que la séparation des variables ne sera pas facile. On procède alors à l’apprentissage du modèle en version linéaire avec recherche de meilleur paramètre C à l’aide d’une grille de valeurs et une optimisation par validation croisée :

On obtient les scores suivant sur l’échantillon d’entraînement de test ainsi que le paramètres C utilisé :

Meilleur paramètre C : 0.03962

Generalization score for linear kernel:	`Train : 0.6933`	`Test : 0.68`

Le score de l'échantillon train montre des valeurs proche de 70% avec des résultats presque aussi bon pour l'échantillon test. On pouvait s'y attendre au vue du nuage de points.

Question 2 :

On procède de même mais avec un noyau polynomial :

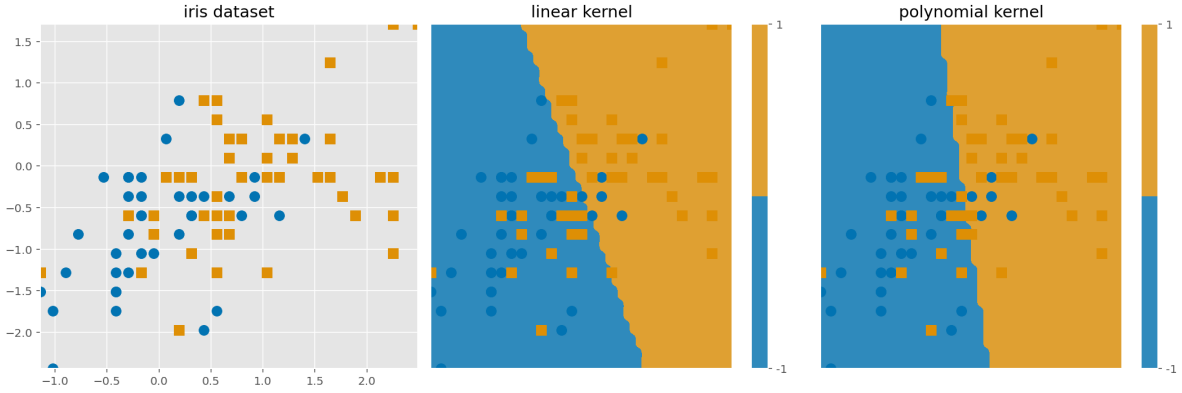
Et les résultats :

Meilleur parmètre C : 0.03962688638701478 ,

Generalization score for polynomial kernel:	Train : 0.72	Test : 0.76

On note une amélioration modérée de la séparation des variables. On "hyperplan courbe" est évidemment plus adapté à la distribution des données mais les points sont tout de même fortement "mélangés" et un séparation complète des variables semble bien compliquée.

Pour conclure la question, l'illustration graphique :

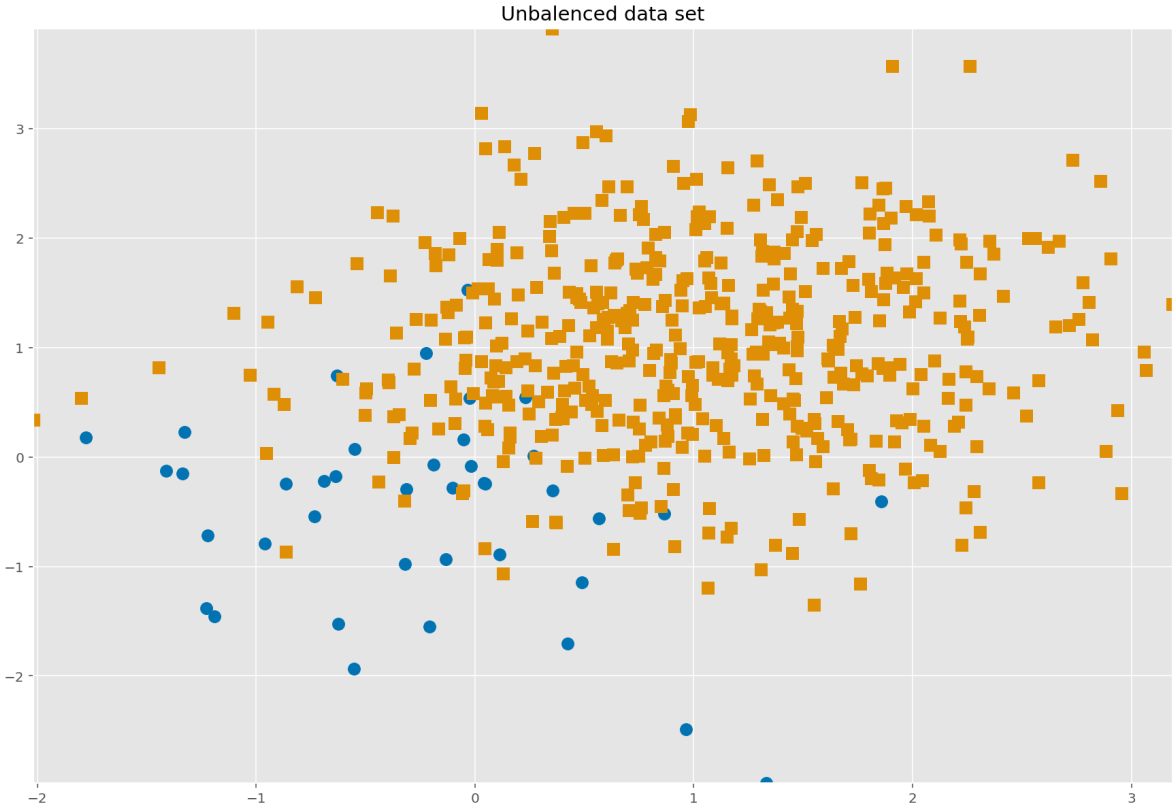


On voit que le gain est faible avec le modèle polynomial.

Question 3 :

La pratique de SVM_GUI n'a montrée aucune interactivité sur mon navigateur (Edge). Mais quelques essais on réussit.

On génére un jeu de données très déséquilibré avec beaucoup plus de points dans une classe que dans l'autre ici (92% vs 8%).



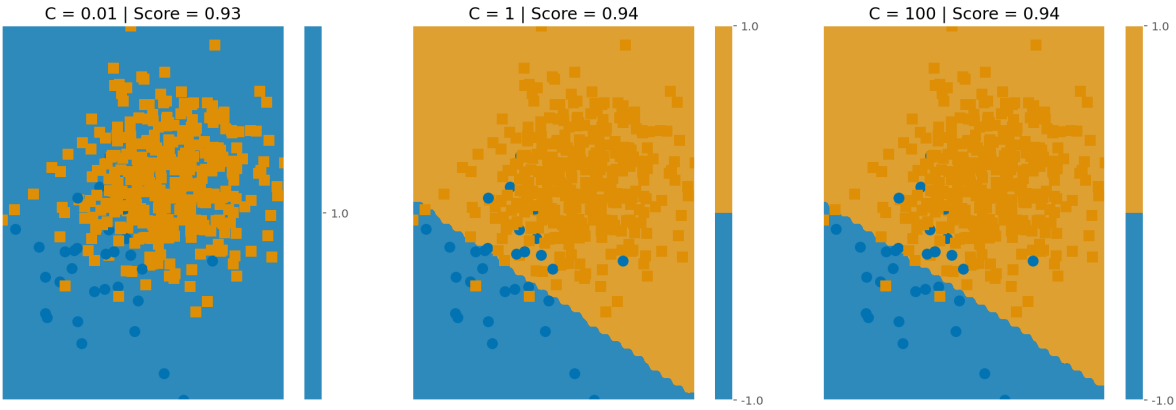
On procède de même que précédemment en paramètres par défaut avec un noyau linéaire :

Et le résultat sur l'échantillon test :

Score : 0.9407

La performance est bonne mais voyons l'influence C sur ce score (arrondi à 10^-2)

On obtient les représentations suivantes :



C contrôle la pénalité sur les erreurs de classification.

Quand C est très petit, le modèle tolère beaucoup d'erreurs pour maximiser la marge. Il préfère une marge large, même si cela signifie mal classer plusieurs points. Autrement dit, avec C = 0.01, le SVM devient très souple : il accepte que des points #soient mal classés pour obtenir une frontière plus "stable" ou "généreuse". Si les classes ne sont pas parfaitement séparables, ou si tu as du bruit, le modèle peut même ignorer complètement la structure réelle des données. comme on le voit à gauche. Si C = 1 : pénalise les erreurs modérément.

Si C = 100 : pénalise fortement les erreurs → le modèle essaie de tout classer parfaitement.

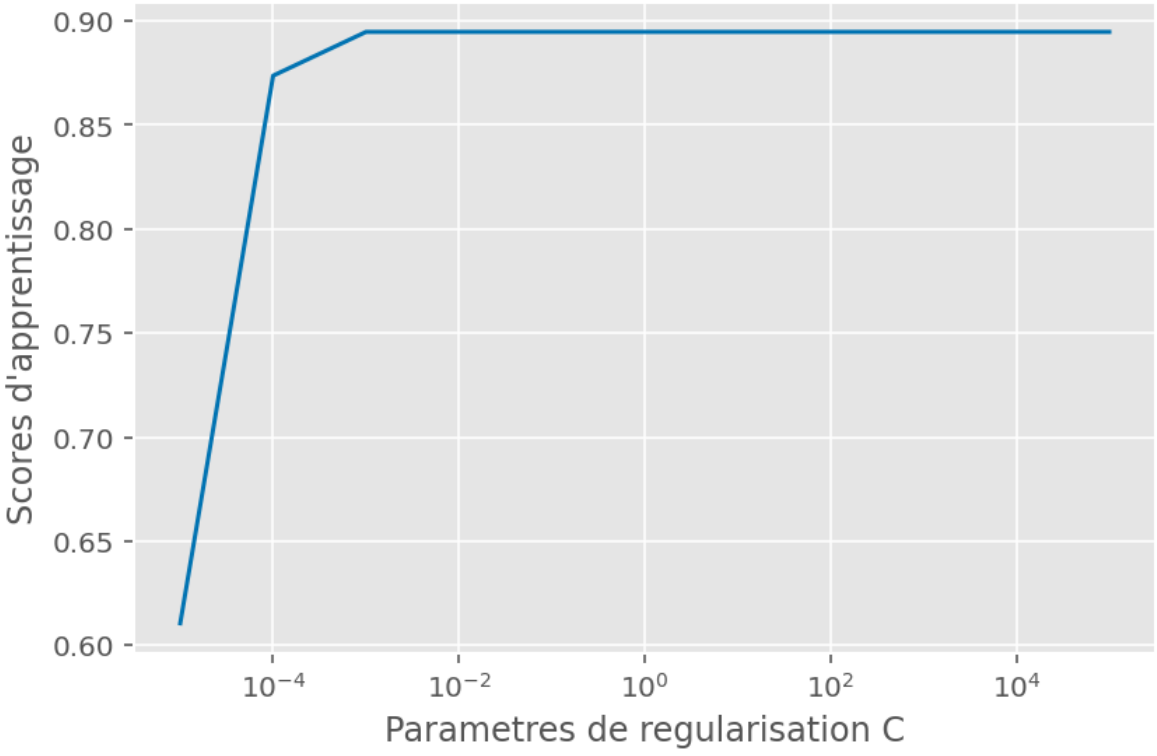
Mais si les données sont déjà bien séparées, alors même un C modéré suffit à tracer une frontière quasi parfaite. Le modèle n'a pas besoin de forcer davantage, donc la frontière reste quasi identique.

Question 4 :

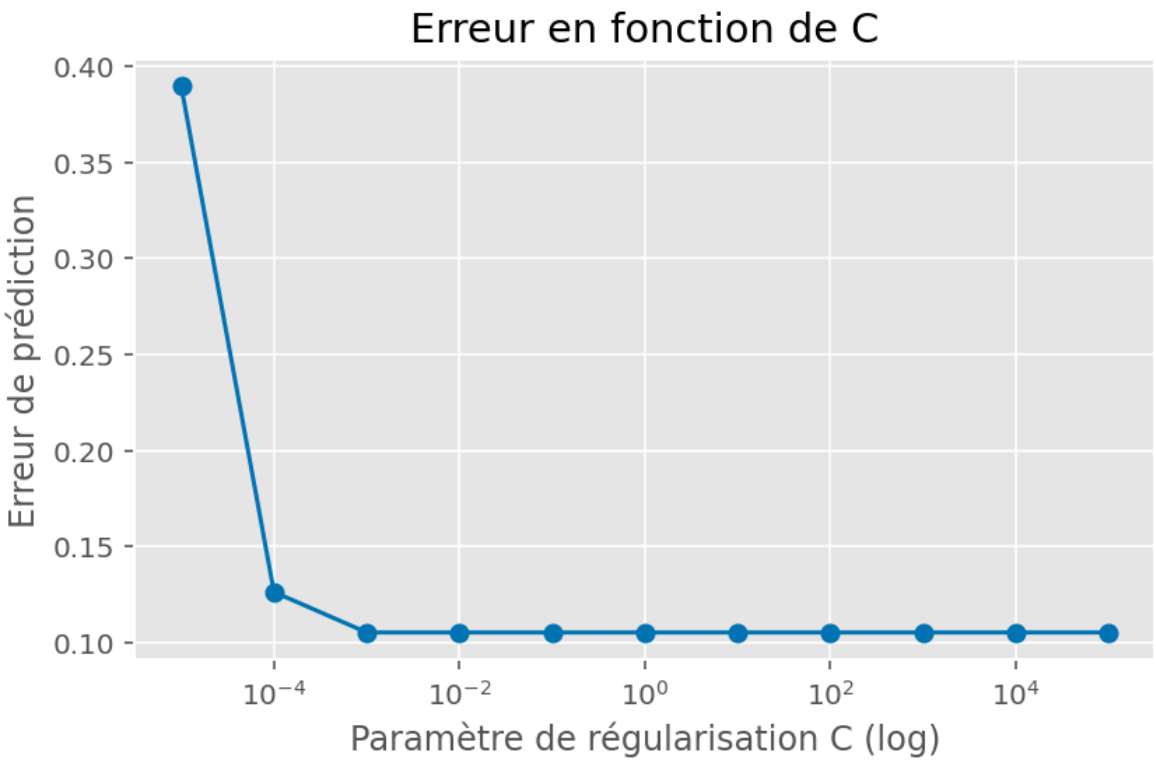
On commence par l'importation des images à traiter. Le fichier et téléchargé, décompressé et le dossier obtenu est déplacé dans le répertoire des scriptes. Le scripte fourni récupère dans X les images et dans Y les noms et selectione uniquement les données Tony Blair et Colin Powell. Voici 12 images tirées de la base de photos :

Ensuite, on procède à la préparation des données. Les 3 canaux RVB des images sont moyennés ce qui conduit à une image en noir et blanc. Puis les données sont centrées réduites et enfin séparées en 2 échantillon Train et Test (50/50).

On réalise des graphique du score et des erreurs en fonction C sur en premier l'échantillon apprentissage.


0.90
0.85
0.80
0.75
0.70
0.65
0.60
Scores d'apprentissage
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
Parametres de regularisation C

ou



Les résultats :

— Linear kernel —

Fitting the classifier to the training set

Best C: 0.001

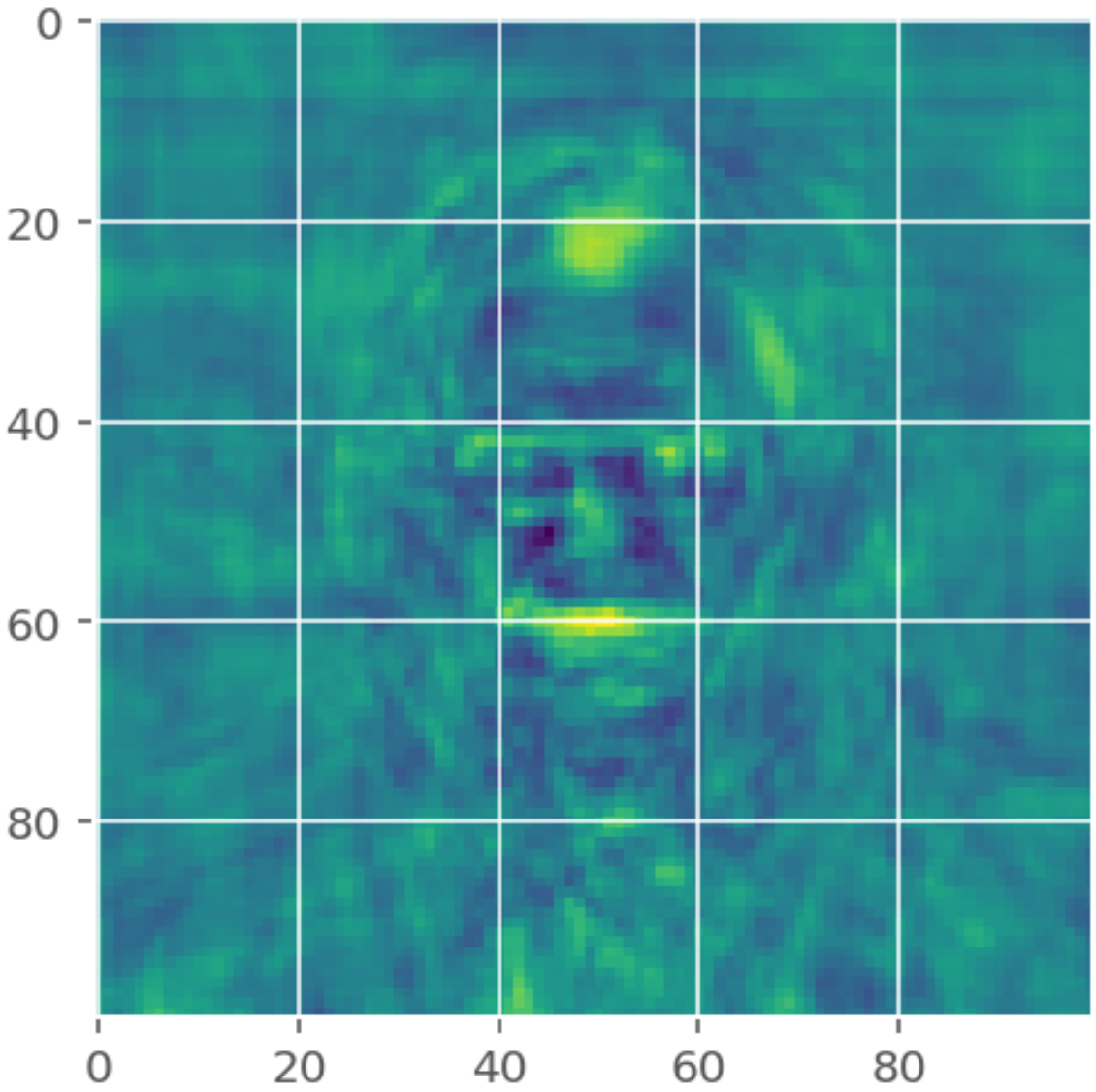
Best score: 0.8947

On constate la présence du “coude” qui donne le meilleur compromis et que :

- C petit → marge large, plus d’erreurs → erreur élevée.
- C grand → modèle rigide, moins d’erreurs (mais attention à l’overfitting).
- Le minimum de la courbe donne le compromis optimal entre biais et variance.

Puis on procède à la prédiction des noms sur l’échantillon test avec le meilleur paramètre C :

Et l'image modèle issue du fit :



Et les résultats de la console :

- Predicting the people names on the testing set
- done in 0.085s
- Chance level : 0.6210526315789474
- Accuracy : 0.8947368421052632

On constate un très bon niveau de prédiction malgré des images par toujours centrée. C'est surprenant pour un débutant en IA.

Question 5 :

On ajoute via des bruits gaussiens au images et on compare les résultats de SVM.

On obtient les résultats suivant :

Generalization score for linear kernel:	Train	Test
Score sans variable de nuisance	1.0	0.90526
Score avec variable de nuisance	1.0	0.50526

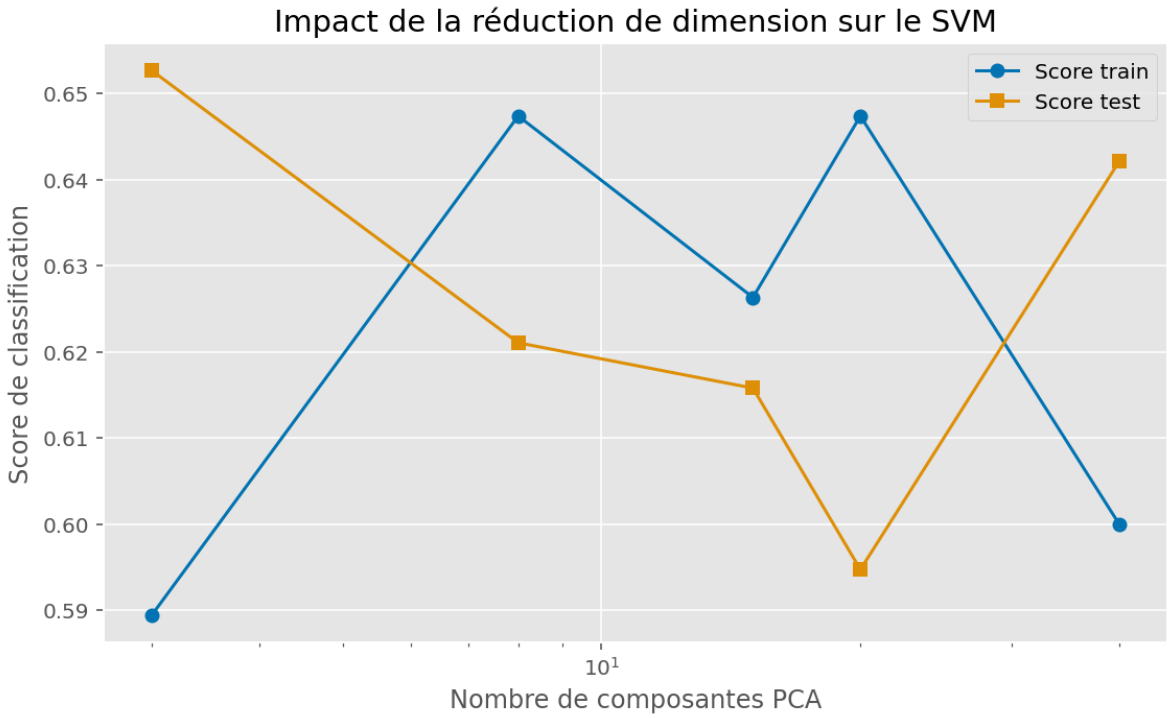
On constate que l'ajout n'a pas d'effet sur l'échantillon d'apprentissage. On peut penser que le fit des images floues peut s'ajuster à ce qu'il voit. Par contre, sur les images inconnu floutées, c'est beaucoup plus difficile de les identifier.

Question 6 :

L'idée est de procéder à une ACP sur les X bruités pour en sortir les composantes principales, les premières devant être informatives et celle de plus haut rang, être du bruit.

Au vue de la question 7, on peut se demander, s'il faut faire l'ACP puis spliter l'échantillon ou faire le split puis 2 ACP pour le train et le test. Mon choix c'est porté sur le premier protocole avec souci de comparer des choses sur les mêmes composantes. Probablement, avec un biais ajouté.

Le graphique avec en fonction de n composantes :



D'abord, sans initialiser de graine, le graphique change à chaque fois. Néanmoins, pour l'échantillon train, on observe une amélioration modérée du score d'apprentissage jusqu'à une certaine valeur de n (CP) puis une baisse des performance. C'est normale, trop de composantes apporte du bruit.

Pour l'échantillon test, globalement le score de reconnaissance diminue avec n qui augmente. Le maximum d'information est donc dans les premières composante. Entre n = 20 et 40 composante, on voit le score remonter. Ils y aurait donc des composantes informatives caché parmi celle de bruit et on peut penser que rapidement les composantes informatives et de bruits sont mélangées.

On note une amélioration modérée dans l'identification des photos sur l'échantillon test qui au mieux est maintenant supérieure à 65% au lieu de 50 %

Question 7 :

Le biais est introduit le centré-réduit avec le split des échantillon. De fait, il ne sont pas totalement indépendant, car normalisés avec des infos connues des 2…