TP3 - Support Vector Machine (SVM)

Guillaume Bernard-Reymond

27 septembre 2024

Le but de ce TP est de mettre en pratique ce type de techniques de classification sur données réelles et simulées au moyen du package scikit-learn (lequel met en œuvre la librairie en C libsvm) et d'apprendre à contrôler les paramètres garantissant leur flexibilité (hyper-paramètres, noyau)

Dans tout le rapport, les résultats seront arrondis à 10^{-4} pour garder une certaine précision tout en restant lisible. On trouvera tout au long du compte-rendu les morceaux de code qui étaient à compléter ainsi que des ajouts personnels.

Première mise en œuvre

Question 1 et 2 :

On sépare d'abord le jeu de données `iris` en deux parties : une pour l'entrainement et l'autre pour le test :

```
X, y = shuffle(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5)
```

J'ai dans un premier temps appliquer une SVM au jeu de données `iris` sans optimisation du choix des paramètres par l'utilisation de la fonction `GridSearchCV` dans le cas linéaire et polynomiale :

Noyau linéaire	0.56
Noyau polynomiale	0.5

Table 1 – Comparaison des scores sur la prédiction des classes 1 et 2 du jeu de données iris sans optimisation

On remarque donc que le choix par défaut du noyau polynomial ne rend pas meilleur la prédiction bien que l'espace de plongement soit supérieur

On peut observer aussi le choix de classification qui a été fait :

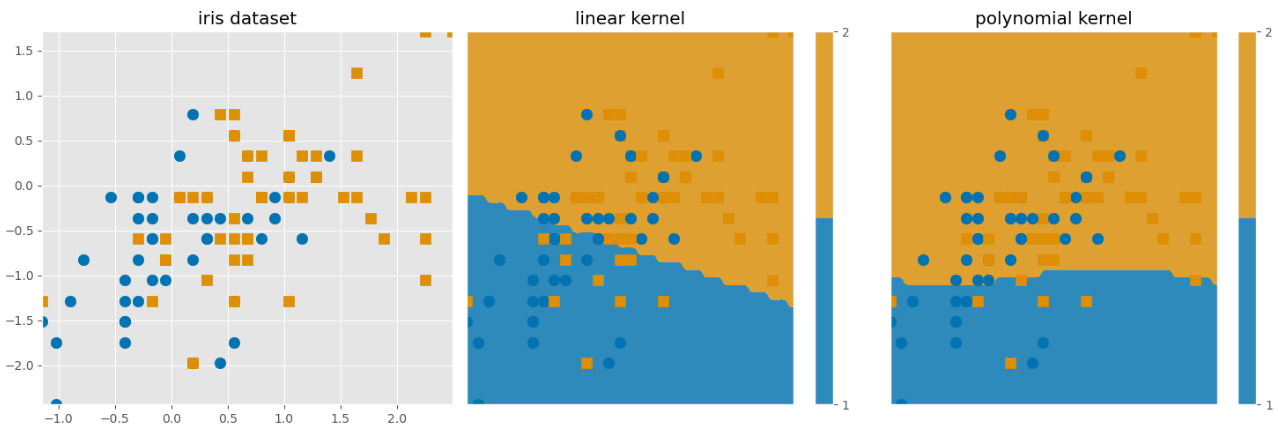

Figure 1 – Comparaison de la classification des iris entre un noyau linéaire et un noyau polynomial avec paramètres par défaut.

Maintenant si l'on choisit d'optimiser les paramètres, on obtient à partir du code ci-dessous les résultats suivants :

```
# fit the model
parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3, 200))}
clf_linear = SVC()
clf_linear_grid = GridSearchCV(clf_linear, parameters, n_jobs=-1)
```

```
clf_linear_grid.fit(X_train,y_train)

#calcul de y_pred mais pas forcement utile ici
y_pred=clf_linear_grid.predict(X_test)

# compute the score
print(clf_linear_grid.best_params_)
score = clf_linear_grid.score(X_test,y_test)
```

Noyau linéaire	0.56
Noyau polynomial	0.62

TABLE 2 – Comparaison des scores sur la prédiction des classes 1 et 2 du jeu de données iris avec optimisation

Le gain d'environ 10,7% par rapport au noyau linéaire n'est pas négligeable, toutefois il faut prendre les temps de calcul qui sont alors plus long car plus de paramètres sont à tester et à valider.

On observe une classification cette fois ci bien différente :

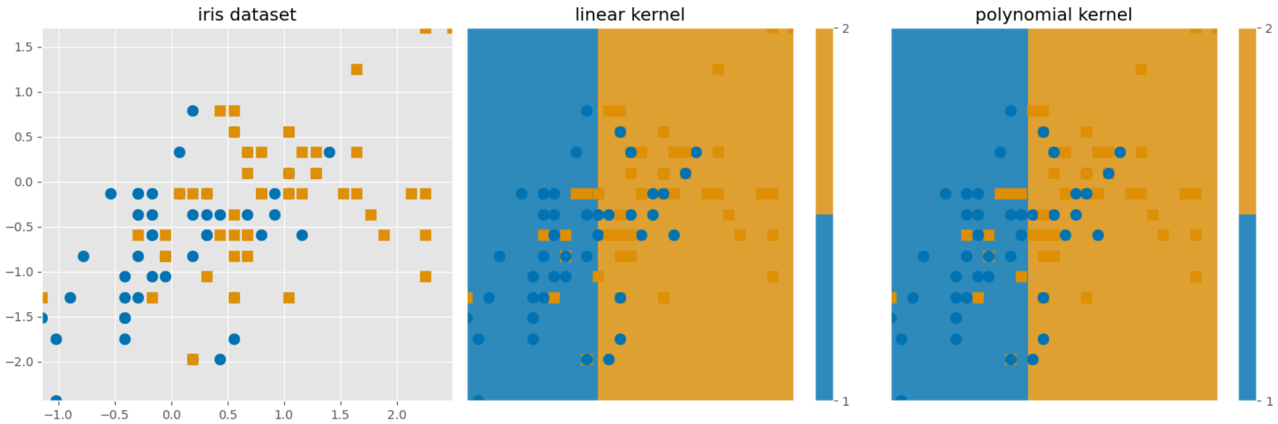

FIGURE 2 – Comparaison de la classification des iris entre un noyau linéaire et un noyau polynomial optimisé.

Dans notre cas, il semble que pour le noyau polynomiale optimisé, un choix linéaire ait été fait.

SVM GUI

Question 3 :

Pour un nuage de points fixé dans des proportions 90% - 10%, nous nous intéressons à l'influence d'un paramètre C appelé "constante de tolérance" et qu'il s'agit d'ajuster pour avoir le pourcentage de classement correct le plus élevé : l'accuracy.

Valeur de C	1	0.1	0.01	0.001	0.0001	0.00001
Accuracy	96	96	96	92	90	90

TABLE 3 – Evolution de l'accuracy en fonction de la valeur de C

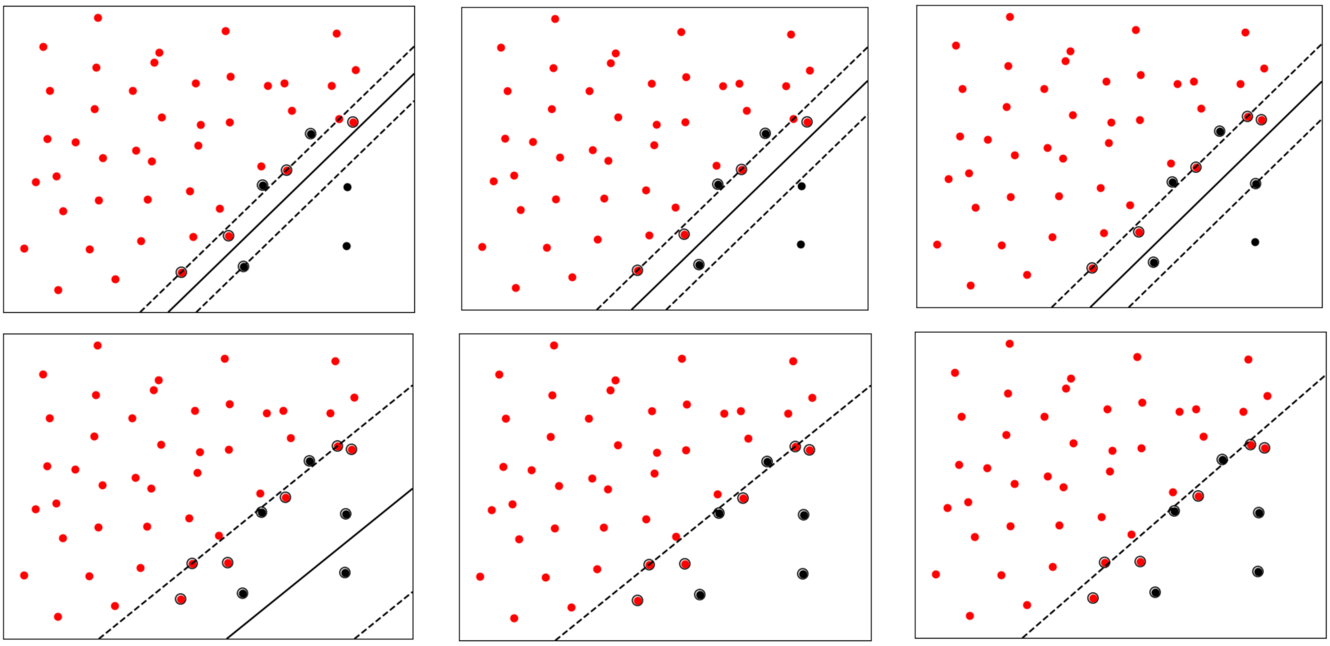

FIGURE 3 – Classification avec un noyau linéaire pour différentes valeurs de C = 1 ; 0.1 ; 0.01 ; 0.001 ; 0.0001 ; 0.00001 de gauche à droite et de haut en bas

On remarque l'accuracy baisse avec la valeur de C. Ceci est tout à fait normal car lorsque C diminue, la marge augmente et le pourcentage de bien classé diminue vu qu'on autorise certains à être dans la marge. D'ailleurs pour $C < 0,001$, tous les points noirs sont mal classés.

Classification de visages

Question 4 :

Le score d'apprentissage augmente avec la valeur de C jusqu'à atteindre un palier à partir de $C = 10^{-3}$ que l'on va donc considérer comme notre meilleur paramètre.

```
# fit a classifier (linear) and test all the Cs
Cs = 10. ** np.arange(-5, 6)
scores = []
for C in Cs:
    clf = SVC(kernel='linear', C=C)
    clf.fit(X_train, y_train)
    scores.append(clf.score(X_train, y_train))
```

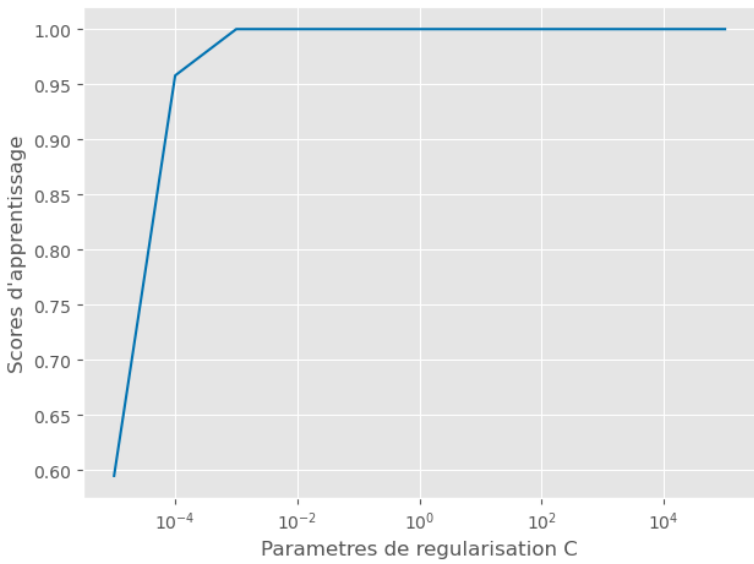

FIGURE 4 – Evolution du score en fonction du paramètre de régularisation C

On obtient alors les résultats suivants pour la valeur de $C = 10^{-3}$:

- Chance level : 0.6211
- Accuracy : 0.9105

```
# predict labels for the X_test images with the best classifier
clf = SVC(kernel='linear', C=Cs[ind])
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
```

On pourrait aussi tracer la courbe de l'erreur par validation croisée sur le paramètre C.

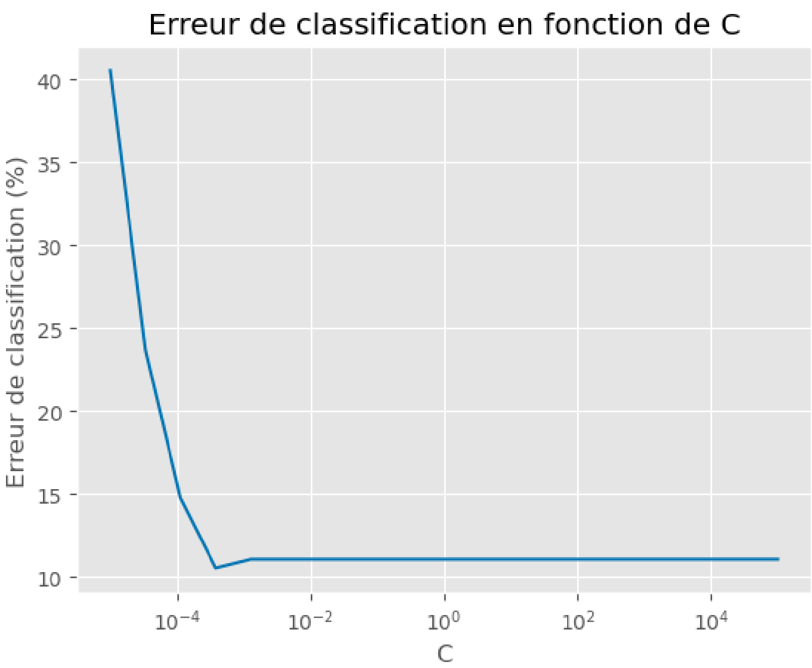

FIGURE 5 – Erreur de prédictions en fonction de C obtenue par validation croisée.

On retrouve sur cette figure 5, la valeur de $C = 10^{-3}$ qui va minimiser l'erreur de prédiction.

```
#courbe de l'erreur par cross validation
from sklearn.model_selection import cross_val_score
err = []

for C in Cs:
    clf = SVC(kernel='linear', C=C)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    err.append((1 - scores.mean())*100)
```

Et qu'est ce que cela donne si on revient aux images ?

FIGURE 6 – Prédiction entre Colin Powell et Tony Blair

On observe en conséquence sur le jeu de données des images une erreur de une prédiction sur les 12 ce qui correspond bien à une Accuracy de $0,91$.

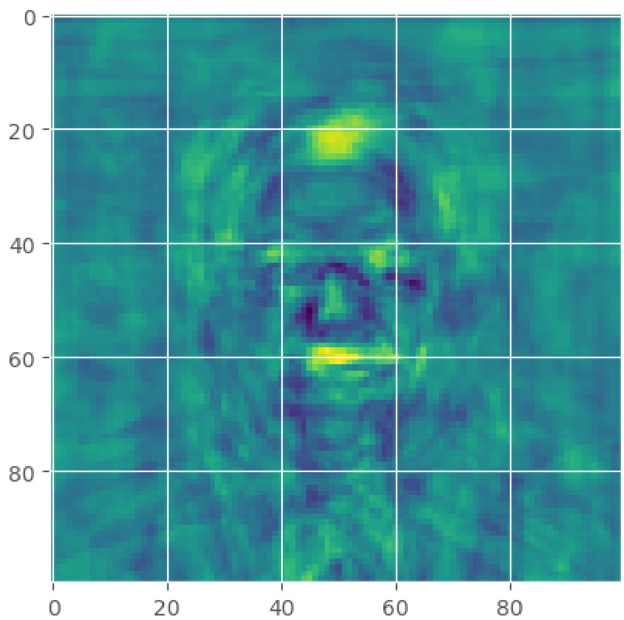

FIGURE 7 – Importance des différents pixels dans la décision

Enfin sur la figure 7, on peut apercevoir les pixels aux couleurs les plus intenses, claires ou foncées. Ce sont ces pixels, ces variables, qui apportent le plus de poids dans la prédiction de classification.

Question 5 : Introduction de nuisance
J'ai décidé de fixer la graine à l'intérieur de la fonction `run_svm_cv`. Ainsi la comparaison se fera sur des jeux de données identiques. De plus dans la partie avec bruit, la ligne de code :

```
X_noisy = X_noisy[np.random.permutation(X.shape[0])]
```

n'est pas valable car on modifie les images mais pas les labels. De plus dans la fonction `run_svm_cv`, le mélange est déjà fait. Avec $\sigma = 1$ on n'observe pas de différence et j'ai donc décidé d'augmenter sigma à 5 qui est déjà une valeur importante car nos données sont déjà centrées et réduites.

```
run_svm_cv(X,y)
run_svm_cv(X_noisy,y)
```

On observe donc une baisse du score. Il faudrait peut-être rajouter plus de variables de bruit pour observer une vraie baisse.

- Score sans variable de nuisance : 0.8947
- Score avec variable de nuisance : 0.8158

Question 6 : score après réduction de dimensions
En diminuant la dimension de l'espace de départ en cherchant les composantes principales par ACP voici les résultats obtenus :

Nombre de composantes	5	10	20	50	100	380
Score	0.5789	0.6421	0.5895	0.7789	0.8105	0.8105

Table 4 – Score obtenu pour des variables bruitées en fonction du nombre de composantes principales

```
n_components = 20  # jouer avec ce parametre
print(f'Score apres reduction de dimension avec {n_components} composantes principales')
pca = PCA(n_components=n_components).fit(X_noisy)
X_pca = pca.transform(X_noisy)
run_svm_cv(X_pca,y)
```

Même en prenant le plus grand nombre de composantes principales possibles, notre score est moins bon. Cela ne sert donc pas à grand chose de d'utiliser cette méthode dans notre étude.