STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER

Master 2 Statistiques et Sciences des Données -BioStats

Compte Rendu TP3
Support Vector Machine (SVM)

MANNEQUIN Jeanne

Année 2024 - 2025

Table des matières

Introduction

Les **SVM** sont des algorithmes d'apprentissage supervisé servant à trouver une frontière de décision optimale entre plusieurs classes. Ils s'utilisent notamment dans les espaces de grande dimension. Leur objectif principal pour obtenir la meilleure prédiction est de maximiser la distance (ou la marge) entre chaque point et l'hyperplan.
Dans ce TP nous allons 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).

Premiers pas

Question 1

Le fichier `script.py` nous donne un exemple de classification avec la fonction `SVM` dont nous nous servons pour classifier la classe 1 contre la classe 2 du data set `iris` (ses deux premières variables) en utilisant un noyau linéaire.
On découpe le jeu de données en 2 sous-ensembles : un ensemble d'apprentissage et un ensemble de test. On laisse donc la moitié du jeu de données de côté pour l'évaluation.
On charge `iris` et des échantillons X et y une première fois et ces échantillons seront utilisés jusqu'à la fin du TP. Pour d'autres échantillons on obtient des résultats différents.

```
# load the iris dataset
iris = datasets.load_iris()
X = iris.data
X = scaler.fit_transform(X)
y = iris.target
X = X[y != 0, :2]
y = y[y != 0]
```

Ensuite on crée et entraîne le classifieur `SVM` avec le noyau linéaire. Puis on en sort les meilleur paramètres et son score pour évaluer la performance du modèle.

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

print("Meilleurs paramètres pour le noyau linéaire : ", clf_linear.best_params_)

# compute the score
train_score = clf_linear.score(X_train, y_train)
test_score = clf_linear.score(X_test, y_test)

print('Generalization score for linear kernel: %s, %s' % (train_score, test_score))
```

On trouve par exemple ici une valeur de C d'environ 0.042 . C est plutôt faible, ce qui veut dire que le modèle accepte plus d'erreurs de classification pour permettre une marge plus large. Cela donnera une frontière plus lisse mais risquerais de faire du sous-apprentissage (en opposition au sur-apprentissage) en cas jeux de données à peu de valeurs (ce n'est pas le cas ici).
On trouve également 0.78 et 0.68 comme meilleures scores pour respectivement l'échantillon d'entraînement et celui de test. Ces scores semblent bon sans être excellents. On constate une perte de 0.1 entre l'échantillon d'entraînement et celui de test, ce qui indique une potentielle baisse de performance sur les valeurs non étudiées. Cela peut également indiquer un overfitting.

Le score de l'échantillon d'entraînement a beau être bon, la différence avec le score de test indique que le modèle, avec ces échantillons initiaux, pourrait faire mieux avec d'autres paramètres.

Question 2

Essayons donc par exemple d'effectuer le même schéma avec un noyau différent, le noyau polynomial, et comparer de les scores obtenus.

	train_score	test_score	Différence
Noyau linéaire	0.78	0.68	0.1
Noyau polynomial	0.76	0.7	0.06

TABLE 1 – Scores pour les noyaux linéaire et polynomial

Dans le tableau 1 on voit que les scores avec le noyau polynomial sont globalement meilleurs, car même si le score de l'échantillon d'apprentissage est plus bas, la différence avec celui de test est encore plus basse, donc pas ou peu d'overfitting !

SVM GUI (optionnel)

Question 3 (bonus)

On fait tourner le code disponible sur le fichier En augmentant le paramètre C la marge se réduit et inversement en diminuant C.

Classification de visages

Dans cette partie, on s'intéresse à la classification de visage. On va donner au modèle des images d'un certain Tony Blair puis on va lui demander de différencier cette personne d'une autre : Colin Powell. Voici en figure 1 les photos données au modèle :

FIGURE 1 – Photos de Tony Blair données au modèle pour entraînement

On regarde ensuite les résultats avec l'échantillon de validation sur la figure 2 :

FIGURE 2 – Résultat des prédictions du modèle SVM par image de Blair et de Powell

Le "chance level", c'est-à-dire la précision que le modèle atteindrait s'il prédisait toujours la classe majoritaire, est ici d'environ 0.62 . C'est la valeur à laquelle on va comparer l'"accuracy" du modèle. L'"accuracy" est, elle, d'environ 0.92 . Le résultat est bon, mais quels sont les paramètres choisis ?

Question 4

En premier lieu, on se propose d'étudier le score d'apprentissage selon C choisi entre 10^{-5} et 10^{5}. Comme on le voit sur cette figure 3 : le score augmente très rapidement jusqu'à atteindre son maximum d'environ 0.92 à $0.001{=}10^{-3}$ (c'est le score que l'on avait trouvé pour la classification de visage en explication de cette partie), puis baisse très légèrement à environ 0.91 jusqu'à 10^{-2} et stagne jusqu'à la fin. On aura donc tendance à choisir un C valant 0.001 pour cet échantillon, c'est ce que le modèle fait automatiquement.

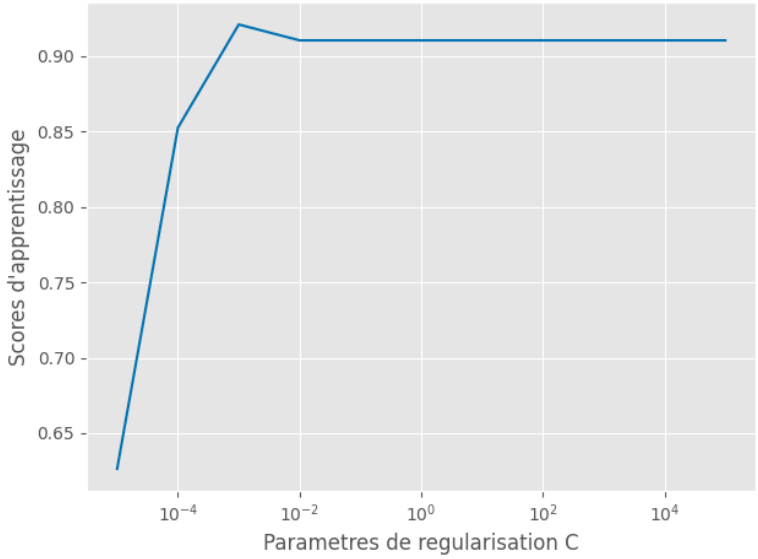

FIGURE 3 – Scores d'apprentissage selon C

Question 5

Intéressons nous maintenant au score de notre modèle si on ajoute du bruit. Logiquement, le score devrait chuter. On lance donc le code suivant sur nos données :

```
n_features = X.shape[1]
# On rajoute des variables de nuisances
sigma = 1
noise = sigma * np.random.randn(n_samples, 300, )
X_noisy = np.concatenate((X, noise), axis=1)
X_noisy = X_noisy[np.random.permutation(X.shape[0])]
```

On applique ensuite le modèle `SVM` sur les données sans nuisances puis avec nuisance pour comparer les scores. On obtient les résultats suivants (arrondis à 10^{-2}) :

	train_score	test_score	Différence
Sans variable de nuisance	1.00	0.92	0.08
Avec variable de nuisance	0.99	0.57	0.42

TABLE 2 – Scores sans et avec variable de nuisance

On constate que le score d'apprentissage ne s'en trouve que très peu diminué mais que l'écart avec le score de validation a drastiquement augmenté. Cette conclusion semble logique, en ajoutant du bruit, le modèle se construit sur des choses "fausses", potentiellement éloignés du réel modèle et donc diminuant le score de validation.

Question 6

Tentons enfin d'améliorer la prédiction du modèle en réduisant la dimension. On fait tourner le code suivant pour 5, 10, 15, 20, 25, 80, 120, et 200 composantes :

```
n_components = 120  # On joue avec ce parametre
pca = PCA(n_components=n_components).fit(X_noisy)
X_reduced = pca.fit_transform(X_noisy)
```

Le code étant très longs à tourner (plus de 10h pour ces 8 valeurs différentes), les scores pour 5, 10, 15, 20, et 25 composantes nous ont gracieusement été donnés par Monsieur FESTOR Quentin, collègue de travail avec un ordinateur puissant et la patience qui nous a manqué. Pour 80, 120 et 200 composantes le code tournait beaucoup plus rapidement. Nous nous permettrons également d'ajouter la variance expliquée pour chaque nombre de composantes. On obtient le tableau suivant (arrondis à 10^{-2}) :

Nb de composantes \ Échantillons	train_score	test_score	Différence	Variance expliquée
5	0.61	0.62	0.01	0.44
10	0.61	0.64	0.03	0.57
15	0.65	0.59	0.06	0.63
20	0.66	0.59	0.07	0.68
25	0.69	0.58	0.11	0.71
80	0.83	0.58	0.25	0.87
120	0.88	0.55	0.33	0.90
200				

TABLE 3 – Scores pour les noyaux linéaire et polynomial

On observe qu'en diminuant la dimension, on diminue le score de l'échantillon d'apprentissage. En revanche, ce qui est intéressant ici c'est que l'écart avec le test de validation s'en trouve bien diminué

également ! Le modèle est donc plus fiable ! Pour ce qui est de la variance explique, elle augmente bien évidemment en augmentant le nombre de composantes. On trouve une variance expliquée de 0.44 pour une dimension de 5 composantes, ce qui est assez faible. En revanche, en prenant par exemple 25 composantes, on arrive à un score sur l'échantillon d'apprentissage de 0.69 (reste acceptable), avec une différence à l'échantillon de test autour de 0.1 (acceptable également) et une variance expliquée de 0.71 (relativement correct).
En conclusion, le nombre de composantes choisi dépendra du résultat désiré : plus bas si on accepte de perdre de l'information, plus haut si on accepte un modèle moins performant. On cherche à maximiser les scores et la variance expliquée, tout en minimisant l'écart entre les score d'apprentissage et de validation. Ici, 25 composantes me semble être un bon compromis.

Conclusion

Ce TP a permis de mettre en pratique les concepts de SVM sur des données réelles et simulées, en explorant l'impact des paramètres sur les performances. Les résultats montrent que le choix du noyau et des paramètres tels que C influencent directement la capacité du modèle, comme observé dans les comparaisons entre noyau linéaire et polynomial. En ajoutant du bruit et en réduisant la dimension des données, nous avons constaté que l'ajustement du modèle est essentiel pour éviter le sur-apprentissage, tout en maximisant les scores d'apprentissage et de validation.