TP n°3 - Support Vector Machine (SVM)

Camille MOTTIER

26 septembre 2024

Objectif :

Le but de ce TP est d'étudier l'utilisation des classifieurs SVM. Nous utiliserons pour cela l'objet `sklearn.svm.SVC` du package `scikit-learn`. Nous travaillerons sur deux bases de données : le dataset `iris` et une base de données composée d'images.

Une introduction aux SVM est présente dans le sujet et ne sera pas reprise dans ce compte-rendu.

Certains extraits de code sont cités dans ce rapport, mais le code utilisé pour obtenir les résultats de ce TP est disponible dans son entièreté sur https://github.com/cmottier/TP3_Apprentissage_stat.

1 Mise en œuvre sur le dataset `iris`

Dans cette partie nous utilisons le dataset `iris` et nous souhaitons classifier la classe 1 contre la classe 2 en utilisant les deux premières variables. Les données sont représentées sur la figure 1. Le jeu de données est séparé en données d'apprentissage et en données de test. Voici les instructions utilisées pour cela :

```
# import 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]

# split train test
X, y = shuffle(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

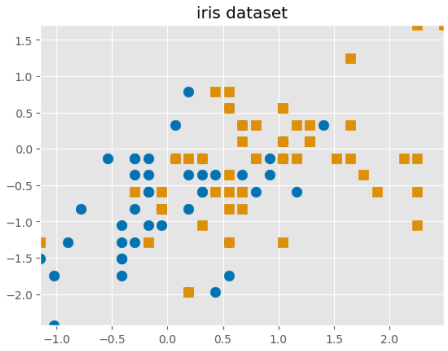


Figure 1 – Classes 1 et 2 du dataset `iris` en fonction des deux premières variables

Question 1.

Nous construisons dans cette question un classifieur SVM à noyau linéaire.

```
# fit the model with linear kernel
clf_lin = SVC(kernel='linear')
clf_lin.fit(X_train, y_train)

# score
print('Generalization score for linear kernel: %s, %s' %
      (clf_lin.score(X_train, y_train),
       clf_lin.score(X_test, y_test)))
```

L'utilisation de ce noyau donne les scores sur les données d'apprentissage et de test suivants :

```
Generalization score for linear kernel: 0.74, 0.68
```

La figure 2 représente le classifieur obtenu par cette méthode.

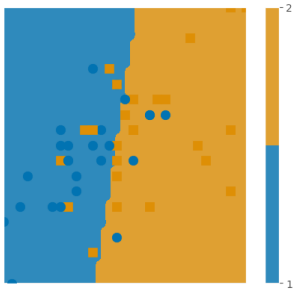


FIGURE 2 – Classifieur SVM à noyau linéaire et données d'apprentissage du dataset `iris`

Question 2.

Nous souhaitons ici voir si l'emploi d'un noyau polynomial nous permet d'obtenir un meilleur classifieur que celui obtenu précédemment. Après avoir adapté le code précédent, nous obtenons la sortie suivante :

```
Generalization score for polynomial kernel: 0.68, 0.52
```

La figure 3 représente le classifieur obtenu avec le noyau polynomial.

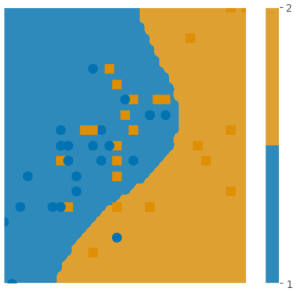


FIGURE 3 – Classifieur SVM à noyau polynomial et données d'apprentissage du dataset `iris`

Nous constatons que le score obtenu avec un noyau polynomial est plus faible que celui obtenu avec un noyau linéaire. Nous préférerons donc le classifieur à noyau linéaire, qui est aussi plus parcimonieux.
La figure 4 permet de visualiser les deux classifieurs sur l'ensemble des données du dataset `iris`.

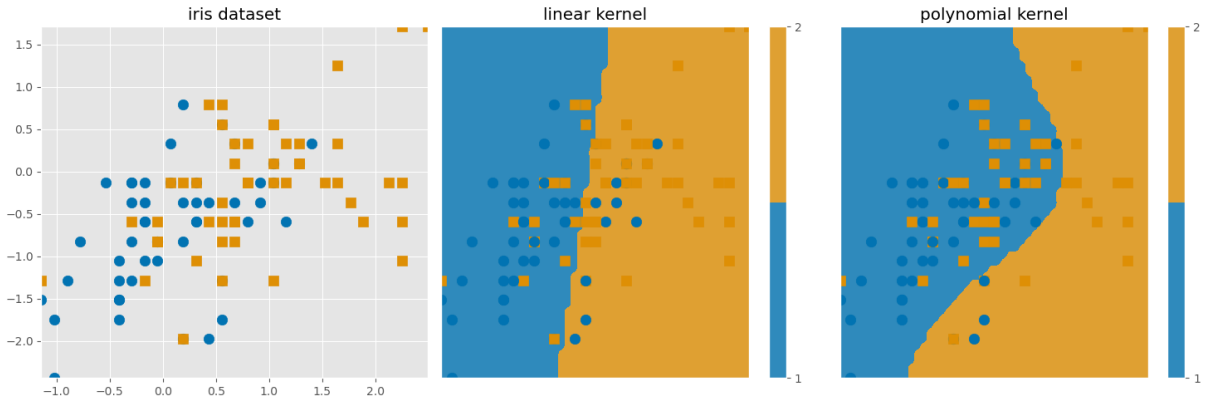


FIGURE 4 – Classifieurs SVM à noyaux linéaire et polynomial et dataset `iris` (apprentissage et test)

Remarque.

Notons que les classifieurs construits aux questions 2 et 3 utilisent les paramètres par défaut de la fonction `sklearn.svm.SVC`. En particulier, la constante de régularisation C est fixée à 1 et le degré du noyau polynomial est fixé à 3. En utilisant la fonction `GridSearchCV`, nous pouvons faire évoluer ces coefficients pour garder le modèle donnant le meilleur score.
Dans le cas du noyau linéaire, en considérant pour C des valeurs de $[10^{-3}, 10^3]$ (`np.logspace(-3, 3, 300)`), nous obtenons les sorties suivantes :

```
{'C': 3.4022997656780714, 'kernel': 'linear'}
Best score : 0.68
```

Le score sur les données de test est le même que celui obtenu à la question 1, malgré une constante C modifiée. Nous n'améliorons donc pas le classifieur à noyau linéaire.
Dans le cas du noyau polynomial, en considérant des degrés entre 1 et 3 et pour C des valeurs de $[10^{-3}, 10^3]$ (`np.logspace(-3, 3, 5)`), nous obtenons les sorties suivantes :

```
{'C': 1.0, 'degree': 1, 'gamma': 10.0, 'kernel': 'poly'}
Best score : 0.7
```

Le score sur les données de test est nettement meilleur que celui obtenu à la question 2. Remarquons cependant que le noyau considéré est de degré 1, ce qui correspond à un noyau linéaire. Nous trouvons alors un score et un classifieur très proches de ceux obtenus dans la question 1 (voir la figure 5).
En conclusion, nous confirmons avec cette remarque le choix d'un noyau linéaire pour construire le classifieur sur les données du dataset `iris`.

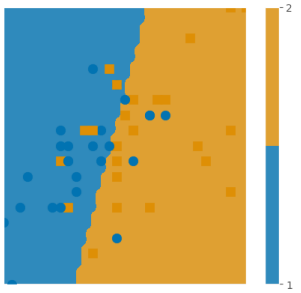
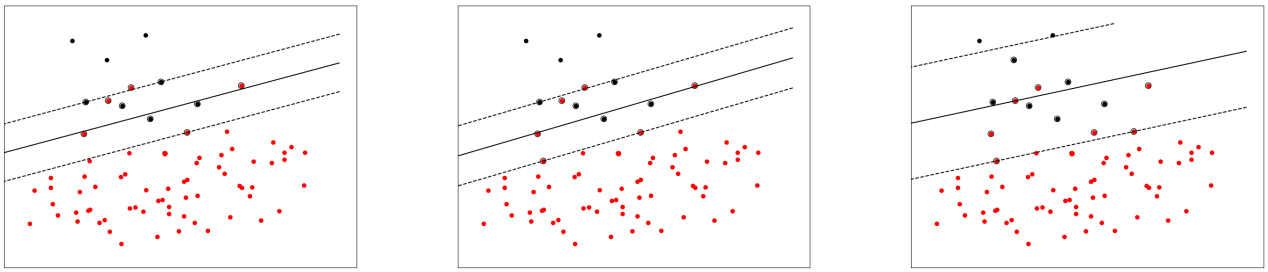


FIGURE 5 – Classifieurs SVM à noyaux polynomial obtenu par `GridSearchCV` (sélectionnant un degré 1) et données d'apprentissage du dataset `iris`

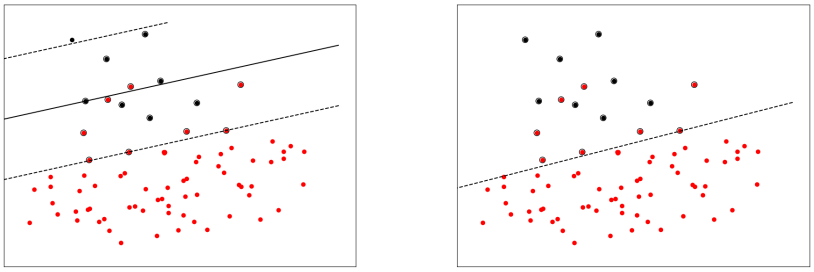
2 SVM GUI

Question 3.

Nous utilisons dans cette question le script `svm_gui.py`, disponible à l'adresse : https://scikit-learn.org/1.2/auto_examples/applications/svm_gui.html.
Il nous permet de visualiser l'impact du choix du paramètre de régularisation C. Sur l'exemple testé (voir figure 6), nous pouvons observer que plus C est petit, plus les marges sont grandes, c'est-à-dire qu'on s'autorise plus d'erreurs. On peut constater que sur la dernière figure, tous les points noirs sont mal classés. Ceci vient vraisemblablement du fait qu'il y a un déséquilibre entre le nombre de points noirs et le nombre de points rouges (les points noirs ont un poids négligeable par rapport à celui des points rouges).



(a) $C = 1$ (b) $C = 0.01$ (c) $C = 0.001$



(d) $C = 0.0005$ (e) $C = 0.0001$

FIGURE 6 – Classifieurs fournis par `svm_gui.py` sur un jeu de données "jouet", en fonction du paramètre de régularisation C

3 Classification de visages

Dans cette partie, nous exploitons une base d'images disponible à l'adresse suivante : http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz. Nous souhaitons classifier à l'aide d'un SVM à noyau linéaire deux types d'images : celles de Tony Blair et celles de Colin Powell. Nous extrayons donc de la base de données les images les concernant et nous séparons nos données en un jeu d'apprentissage et un jeu de test.

Question 4.

Nous souhaitons observer l'influence du paramètre de régularisation C sur la qualité du classifieur. Pour cela, nous construisons une grille de classifieurs pour des valeurs $C = 10^k$, $k \in \{-5, \dots, 6\}$ et observons l'erreur de prédiction en fonction de C.

```
# fit a classifier (linear) and test all the Cs
Cs = 10. ** np.arange(-5, 6)
error = []
for C in Cs:
```

```
    clf = SVC(kernel='linear', C=C)
    clf.fit(X_train, y_train)
    error.append(1-clf.score(X_test, y_test))
```

La figure 7 représente les résultats obtenus.

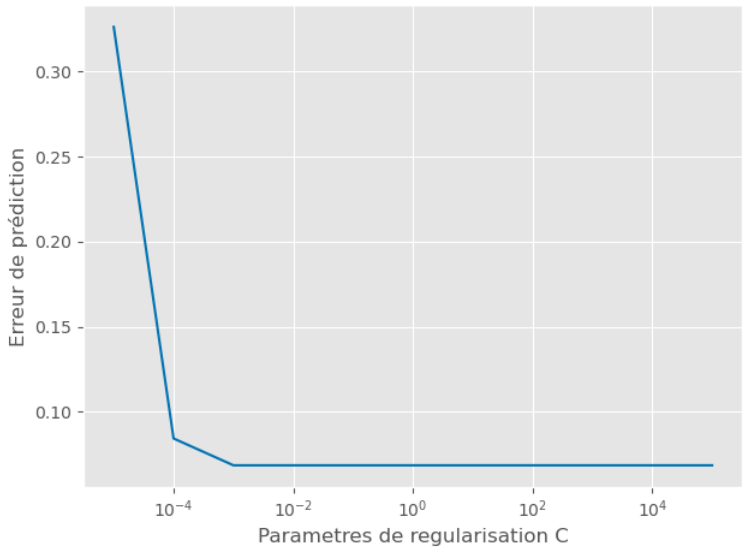


FIGURE 7 – Erreur de prédiction en fonction du paramètre de régularisation pour des classifieurs SVM à noyau linéaire, sur une base de donnée d'images

Nous constatons que l'erreur est élevée lorsque C est très petit (10^{-5}), diminue jusqu'à $C = 10^{-3}$ puis est constante minimale. Ce résultat est cohérent car lorsque C est trop petit, nous ne pénalisons que très peu les mauvais classements, donc ceux-ci peuvent être nombreux.
Nous obtenons pour $C = 10^{-3}$:

```
Best C: 0.001
Best accuracy: 0.9315789473684211
Predicting the people names on the testing set
Chance level : 0.6210526315789474
```

Le score est élevé, en comparaison à celui obtenu en prédisant toujours la classe majoritaire. Ainsi, on peut dire que le classifieur optimal est de bonne qualité. La figure 8 illustre les résultats donnés par la prédiction sur un certain nombre d'images.
Nous pouvons aussi observer quels sont les pixels (ie les variables) qui ont joué un rôle prépondérant, en visualisant les poids qui leur ont été associé. C'est ce que montre la figure 9. Nous devinons l'importance des cheveux, des sourcils et de la bouche pour la prédiction.

Question 5.

Montrons que le score de prédiction est sensible au bruit. Pour cela, nous ajoutons des variables de bruit et recommençons la prédiction. Nous considérons donc le code suivant :

```
# Ajout 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_noisy.shape[1])] # correction du code fourni
run_svm_cv(X_noisy,y)
```

Le bruitage consiste à ajouter 300 variables (pixels) de bruit, suivant une loi normale centrée réduite, et à les mélanger aux autres. Nous obtenons alors les valeurs suivantes avec et sans nuisances :

FIGURE 8 – Prédictions faites par le classificateur optimal sur un échantillon d'images

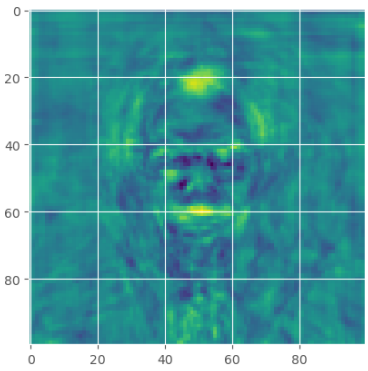


FIGURE 9 – Poids des différentes variables dans le classifieur optimal obtenu

```
Score sans variable de nuisance
Generalization score for linear kernel: 1.0, 0.9315789473684211

Score avec variable de nuisance
Generalization score for linear kernel: 1.0, 0.9105263157894737
```

Nous constatons alors que, malgré les variables de bruit ajoutées, le classifieur est toujours d'excellente qualité. On peut supposer que cela vient du fait que nous avons ajouté seulement 300 variables de bruit, avec un écart-type de 1, à 10000 déjà existantes. En bruitant cette fois-ci avec 1000 variables supplémentaires d'écart-type égal à 5, nous voyons se dessiner une baisse de performance, qui reste cependant légère :

```
Score avec variable de nuisance
```

```
Generalization score for linear kernel: 1.0, 0.8052631578947368
```

Il serait peut-être intéressant de chercher à bruiter les variables ayant un poids fort dans la prédiction afin de visualiser l'effet d'un bruit sur ces variables. La performance devrait alors chuter davantage.

Question 6.

Nous cherchons à améliorer la prédiction obtenue dans la question précédente à l'aide d'une réduction de dimension par analyse à composante principale. Voici le code utilisé pour cela :

```
print("Score apres reduction de dimension")
n_components = 50  # nombre de composantes
X_red = PCA(n_components=n_components).fit_transform(X_noisy)
run_svm_cv(X_red,y)
```

Nous obtenons alors :

```
Score apres reduction de dimension
Generalization score for linear kernel: 0.9157894736842105, 0.7736842105263158
```

Nous remarquons que le score obtenu après l'ACP est moins bon que celui d'origine, ce qui n'est pas très convaincant. Notons qu'en augmentant le nombre de composantes, le score obtenu reste similaire. Il faudrait observer ce qu'il se passe en diminuant le nombre de composantes, mais ma machine n'arrive pas au bout des calculs dans ces cas la.