Rapport TP2 SVM

Mathieu Le Séac'h

1)

On cherche à évaluer les performances d'un classificateur linéaire sur le jeu de données `iris` fourni par `scikit-learn`.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

from lib.svm_source import *
from sklearn import svm
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from time import time

scaler = StandardScaler()

import warnings
warnings.filterwarnings("ignore")

plt.style.use('ggplot')

# load 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_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5, random_state=99)

# fit the model
Cs = list(np.logspace(-3, 3, 200))
gammas = 10. ** np.arange(1, 2)

parameters = {'kernel': ['linear'], 'C': Cs, 'gamma': gammas}

clf_linear = GridSearchCV(SVC(), parameters, n_jobs=-1).fit(X_train, y_train)

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

```
{'C': 0.22478058335487253, 'gamma': 10.0, 'kernel': 'linear'}
Generalization score for linear kernel: 0.76, 0.68
```

On obtient un score de 0.78 sur les données d'apprentissage et de 0.66 sur les données de test.

2)

On cherche à comparer les performances d'un classificateur linéaire avec un classificateur polynomial pour le même jeu de données.

```
# fit the model
Cs = list(np.logspace(-3, 3, 20))
gammas = 10. ** np.arange(1, 2)
degrees = np.r_[1, 2, 3]

parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree': degrees}

clf_poly = GridSearchCV(SVC(), parameters, n_jobs=-1).fit(X_train, y_train)
```

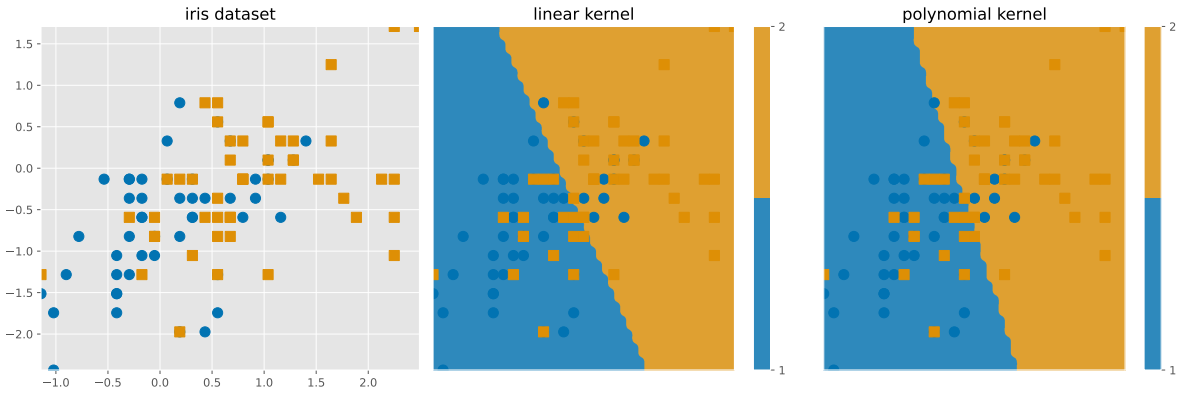
```
# compute the score

print(clf_poly.best_params_)
print('Generalization score for polynomial kernel: %s, %s' %
      (clf_poly.score(X_train, y_train),
       clf_poly.score(X_test, y_test)))
```

```
{'C': 0.0379269019073225, 'degree': 1, 'gamma': 10.0, 'kernel': 'poly'}
Generalization score for polynomial kernel: 0.76, 0.68
```

On constate qu'il y a peu de différence de performance entre le classificateur linéaire et polynomial, ceci s'explique, car le meilleur degré du polynôme trouvé par cross-validation est 1. On se retrouve donc dans le cas d'un classificateur linéaire. Cependant on remarque que l'apprentissage est beaucoup plus long pour le cas polynomial (instantané pour le cas linéaire et quelques secondes pour le cas polynomial malgré le fait que l'espace des paramètres fourni à `GridSearchCV` soit beaucoup plus grand dans le cas linéaire).

```
# display your results using frontiere
def f_linear(xx):
    """Classifier: needed to avoid warning due to shape issues"""
    return clf_linear.predict(xx.reshape(1, -1))

def f_poly(xx):
    """Classifier: needed to avoid warning due to shape issues"""
    return clf_poly.predict(xx.reshape(1, -1))

plt.ion()
plt.figure(figsize=(15, 5))
plt.subplot(131)
plot_2d(X, y)
plt.title("iris dataset")

plt.subplot(132)
frontiere(f_linear, X, y)
plt.title("linear kernel")

plt.subplot(133)
frontiere(f_poly, X, y)

plt.title("polynomial kernel")
plt.tight_layout()
```

```
plt.draw()
```



3)

On cherche à montrer l'influence du paramètre de régularisation C sur les performances.

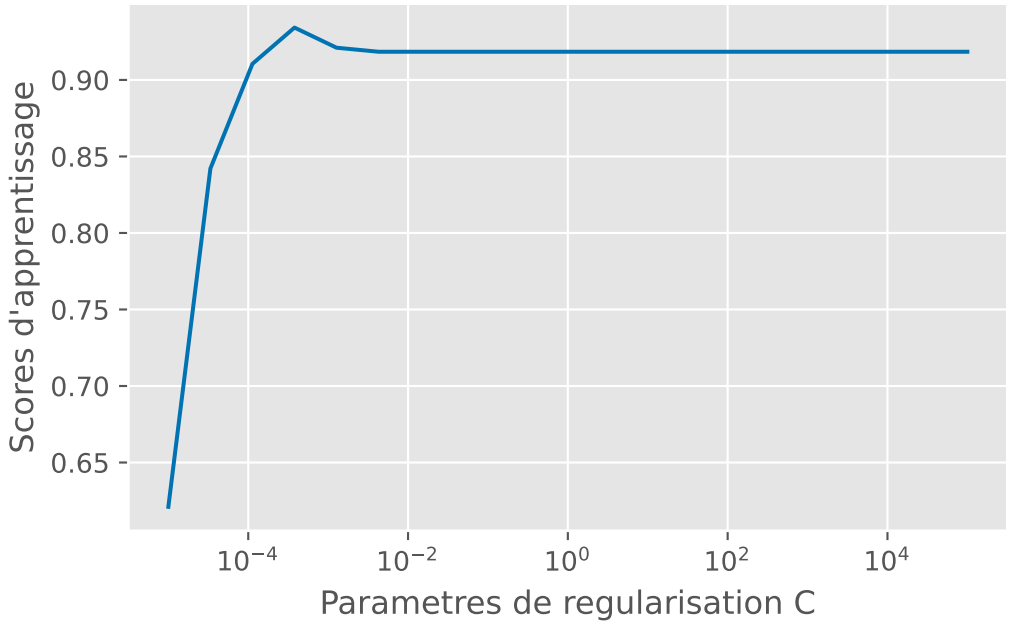
```
# Download the data and unzip; then load it as numpy arrays
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4,
                              color=True, funneled=False, slice_=None,
                              download_if_missing=True)

# introspect the images arrays to find the shapes (for plotting)
images = lfw_people.images

n_samples, h, w, n_colors = images.shape

# the label to predict is the id of the person
target_names = lfw_people.target_names.tolist()

# Pick a pair to classify such as
names = ['Tony Blair', 'Colin Powell']

idx0 = (lfw_people.target == target_names.index(names[0]))
idx1 = (lfw_people.target == target_names.index(names[1]))
images = np.r_[images[idx0], images[idx1]]
n_samples = images.shape[0]
y = np.r_[np.zeros(np.sum(idx0)), np.ones(np.sum(idx1))].astype(int)
```

```
# plot a sample set of the data
plot_gallery(images, np.arange(12))
plt.show()

# Extract features

# features using only illuminations
X = (np.mean(images, axis=3)).reshape(n_samples, -1)

# # or compute features using colors (3 times more features)
# X = images.copy().reshape(n_samples, -1)

# Scale features
X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)

# Q3
print("--- Linear kernel ---")
print("Fitting the classifier to the training set")
t0 = time()

# fit a classifier (linear) and test all the Cs
parameters = {
    'kernel': ['linear'],
    'C': np.logspace(-5, 5, 20),
}

clf = GridSearchCV(SVC(), parameters).fit(X, y)
cv_results = clf.cv_results_

scores = cv_results["mean_test_score"]
Cs = cv_results["param_C"].data

ind = np.argmax(scores)
print("Best C: {}".format(Cs[ind]))

plt.figure()
plt.plot(Cs, scores)
plt.xlabel("Parametres de regularisation C")
plt.ylabel("Scores d'apprentissage")
plt.xscale("log")
```

```
plt.tight_layout()
plt.show()
print("Best score: {}".format(np.max(scores)))

print("Predicting the people names on the testing set")
t0 = time()
```


```
--- Linear kernel ---
Fitting the classifier to the training set
Best C: 0.000379269019073225
```

```
Best score: 0.9342105263157896
Predicting the people names on the testing set
```



On constate que le score d'apprentissage est très mauvais pour C proche de `1e5` puis qu'il atteint son maximum à $C \approx 0.00038$, avant de redescendre et de stagner.

4)

On cherche à montrer l'impacte des variables de nuisance sur la performance des estimateurs.

```python
# Q4
def run_svm_cv(_X, _y, pca=False):
    _indices = np.random.permutation(_X.shape[0])
    _train_idx, _test_idx = _indices[:_X.shape[0] // 2], _indices[_X.shape[0] // 2:]
    _X_train, _X_test = _X[_train_idx, :], _X[_test_idx, :]
    _y_train, _y_test = _y[_train_idx], _y[_test_idx]

    if pca:
        _parameters = {
            'pca__svd_solver': ["randomized"],
            'pca__n_components': list(np.linspace(100, 300, 5, dtype=int)),
```

```
            'svc__kernel': ["linear"],
            'svc__C': list(np.logspace(-3, 3, 5)),
        }

        _pipeline = Pipeline([("pca", PCA()), ("svc", SVC())])
        _clf_linear = GridSearchCV(_pipeline, _parameters)
        _clf_linear.fit(_X_train, _y_train)

    else:
        _parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3, 5)),
}
        _svr = svm.SVC()
        _clf_linear = GridSearchCV(_svr, _parameters)
        _clf_linear.fit(_X_train, _y_train)

    print(f'{_clf_linear.best_params_=}')
    print('Generalization score for linear kernel: %s, %s \n' %
          (_clf_linear.score(_X_train, _y_train), _clf_linear.score(_X_test, _y_test)))

print("Score sans variable de nuisance")

run_svm_cv(X, y)

print("Score avec variable de nuisance")
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])]

run_svm_cv(X_noisy, y)
```

```
Score sans variable de nuisance
_clf_linear.best_params_={'C': 0.03162277660168379, 'kernel': 'linear'}
Generalization score for linear kernel: 1.0, 0.8947368421052632

Score avec variable de nuisance
_clf_linear.best_params_={'C': 0.001, 'kernel': 'linear'}
Generalization score for linear kernel: 1.0, 0.5210526315789473
```

On constate ici que le score chute drastiquement dès qu'on ajoute des variables de nuisance.

5)

On cherche à enlever l'effet des variables de nuisance en utilisant une Analyse par Composantes Principales (PCA).

```
print("Score apres reduction de dimension")
run_svm_cv(X_noisy, y, pca=True)
```

```
Score apres reduction de dimension
_clf_linear.best_params_={'pca__n_components': 100, 'pca__svd_solver': 'randomized', 'svc__C
Generalization score for linear kernel: 0.868421052631579, 0.5684210526315789
```

On constate que le score ne s'améliore pas avec une PCA. Par ailleurs, pour une raison inconnue lorsque j'ai essayé de faire une PCA avec moins de 50 composantes, l'étape d'apprentissage du SVM linéaire ne s'est pas terminé en temps raisonnable (j'ai interrompu le calcul au bout de 5 minutes).