Compte rendu du TP SVM

Fabian Condamy et Samy M'Rad

L'objectif de ce TP est de se familiariser avec la méthode de classification dite de Support Vector Machine (SVM). Dans la suite, on implémentera cette technique sur différents jeux de données réels et simulés grâce au package scikit-learn de Python. On se concentrera principalement sur la compréhension et la maîtrise des principaux paramètres afin d'en ajuster la flexibilité et d'en évaluer l'impact sur les performances.

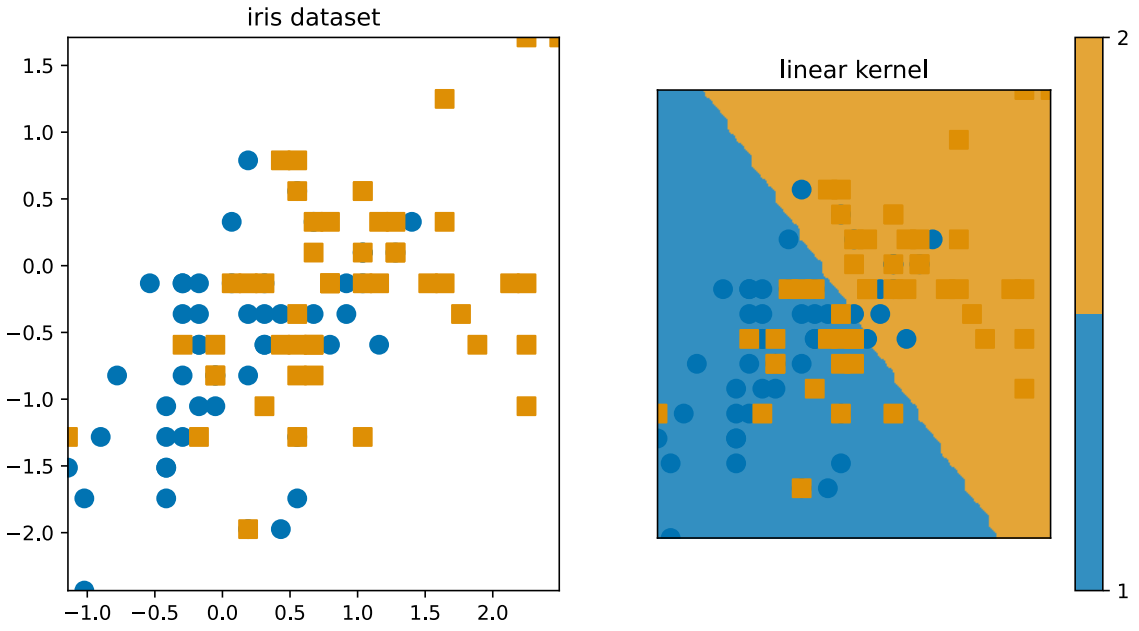
Question 1

```
import sys
sys.path.append("scripts_python") # pour aller chercher la fonction de svm_source
dans le sous dossier
from sklearn import svm
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from svm_source import *
from time import time

scaler = StandardScaler()

# Chargement du jeu de données
iris = datasets.load_iris()
X = iris.data
X = scaler.fit_transform(X)
y = iris.target
X = X[y != 0, :2]
y = y[y != 0]

# Fonction train_test_split du package sklearn
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

# Réalisation du SVM linéaire
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train) # fitting sur la partie train
y_pred_linear = svm_linear.predict(X_test)
```

```
score_linear = svm_linear.score(X_test, y_test)
print('Score du modèle linéaire : %s' % score_linear)

# Fonction de décision pour le traçage
def f_linear(xx):
    return svm_linear.predict(xx.reshape(1, -1))

# Plot
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")
```

```
Score du modèle linéaire : 0.64
```

```
Text(0.5, 1.0, 'linear kernel')
```



Pour cet aléa précis, on trouve un score de 0,64, ce qui est faible. Comme on peut le voir sur le jeu de données iris, les deux classes sont bien mélangées, ce qui peut expliquer ce score médiocre.

```
# Définition des hyperparamètres à tester
parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3, 200))}

# Créer le modèle SVM (objet vide que l'on complétera par la suite)
svm = SVC()

# fonction GridSearchCV de sklearn pour trouver le meilleur C
svm_linear_opt = GridSearchCV(svm, parameters, cv=5)  # cv=5 : validation croisée
5 fois
svm_linear_opt.fit(X_train, y_train)

# Affichage du score
print("Score généralisé pour le noyau linéaire : Train : %.3f | Test : %.3f" %
      (svm_linear_opt.score(X_train, y_train),
       svm_linear_opt.score(X_test, y_test)))
```

```
Score généralisé pour le noyau linéaire : Train : 0.707 | Test : 0.680
```

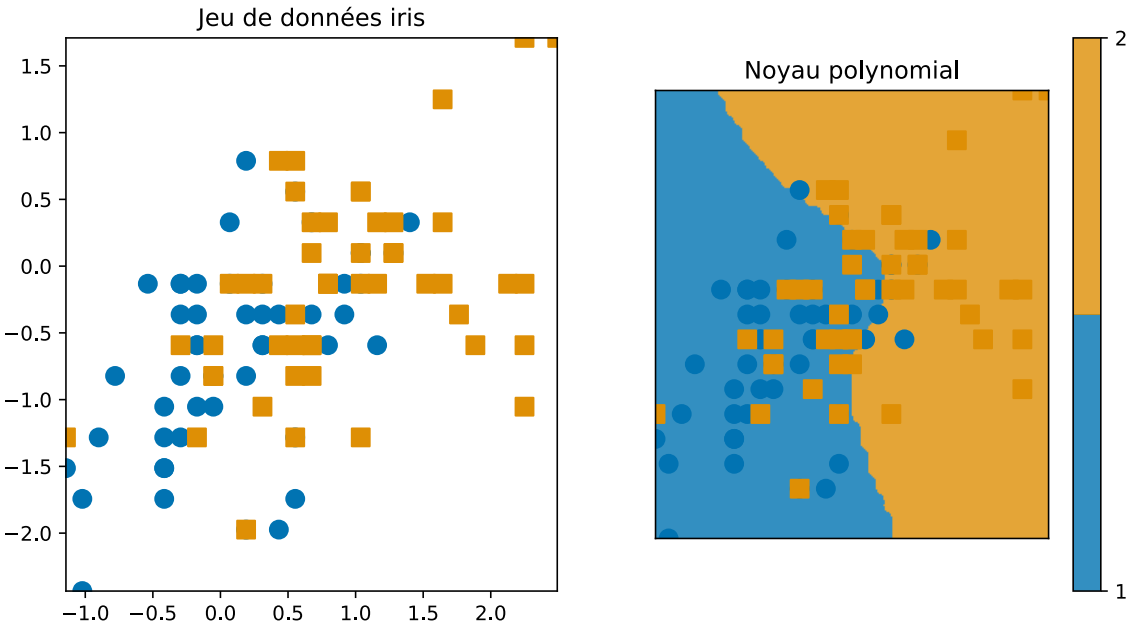
Pour le score généralisé, on trouve une performance de 0,707 pour la partie d'apprentissage et 0,680 pour la partie de test. Ainsi, même en essayant d'optimiser le noyau linéaire, la précision est faible. On va maintenant voir si une méthode polynomiale peut améliorer ce score.

Question 2

```
# On fait la même chose que précédemment mais avec un svm polynomial
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

svm_poly = SVC(kernel='poly')
svm_poly.fit(X_train, y_train)
y_pred_poly = svm_poly.predict(X_test)
score_poly = svm_poly.score(X_test, y_test)
print('Score du modèle linéaire : %s' % score_poly)

def f_poly(xx):
    return svm_poly.predict(xx.reshape(1, -1))

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

plt.subplot(132)
frontiere(f_poly, X, y)
plt.title("Noyau polynomial")
```

```
Score du modèle linéaire : 0.6
```

```
Text(0.5, 1.0, 'Noyau polynomial')
```



Pour un noyau polynomial et cet aléa, on trouve un score de 0,6, ce qui est encore plus faible que pour le noyau linéaire. Ce résultat peut sembler assez paradoxal de prime abord, mais il semble en réalité logique au vu de la structure des données, le modèle a dû certainement overfitter.

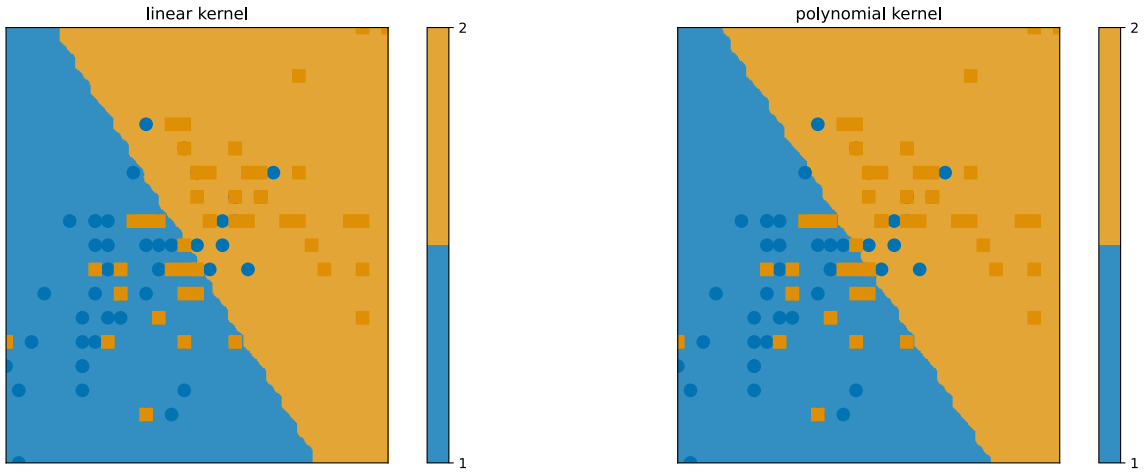
```
# Paramètres que l'on va tester pour trouver les meilleurs
Cs = list(np.logspace(-3, 3, 5))
gammas = 10. ** np.arange(-2, 2)
degrees = np.r_[1, 2, 3]
parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree': degrees}

# idem que précédemment
svm = SVC()

svm_poly_opt = GridSearchCV(svm, parameters, cv=5)
svm_poly_opt.fit(X_train, y_train)

print("Best parameters:", svm_poly_opt.best_params_)

print("Score généralisé pour le noyau polynomial : Train : %.3f | Test : %.3f"
%
      (svm_poly_opt.score(X_train, y_train),
       svm_poly_opt.score(X_test, y_test)))
```

```
Best parameters: {'C': 0.03162277660168379, 'degree': 1, 'gamma': 1.0, 'kernel':
'poly'}
Score généralisé pour le noyau polynomial : Train : 0.707 | Test : 0.640
```

Pour les paramètres généralisés trouvés (c = 0,03 ; degré = 1 et gamma = 1), on trouve un score de 0,707 pour la partie d'apprentissage et 0,64 pour le test. Ainsi, on a le même score généralisé que pour le noyau linéaire pour la partie apprentissage, mais lorsqu'on l'applique à la partie de test, on perd plus en précision que pour le modèle linéaire.

On obtient les graphes suivants, qui sont donc identiques car le polynôme optimal est de degré 1 :

```
# Affichage pour comparaison
def f_linear(xx):
    return svm_linear_opt.predict(xx.reshape(1, -1))

def f_poly(xx):
    return svm_poly_opt.predict(xx.reshape(1, -1))

plt.ion()
plt.figure(figsize=(15, 5))

plt.subplot(121)
frontiere(f_linear, X, y)
plt.title("linear kernel")

plt.subplot(122)
frontiere(f_poly, X, y)

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



Ainsi, ces 2 classes du jeu de données iris semblent très difficiles à séparer, la méthode de SVM est peu adaptée ici.

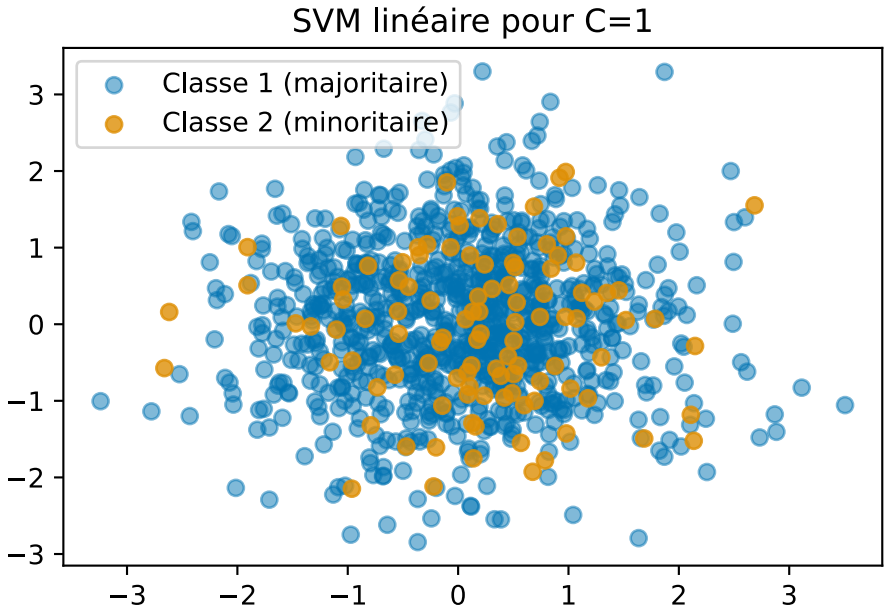
Question 3 (facultative)

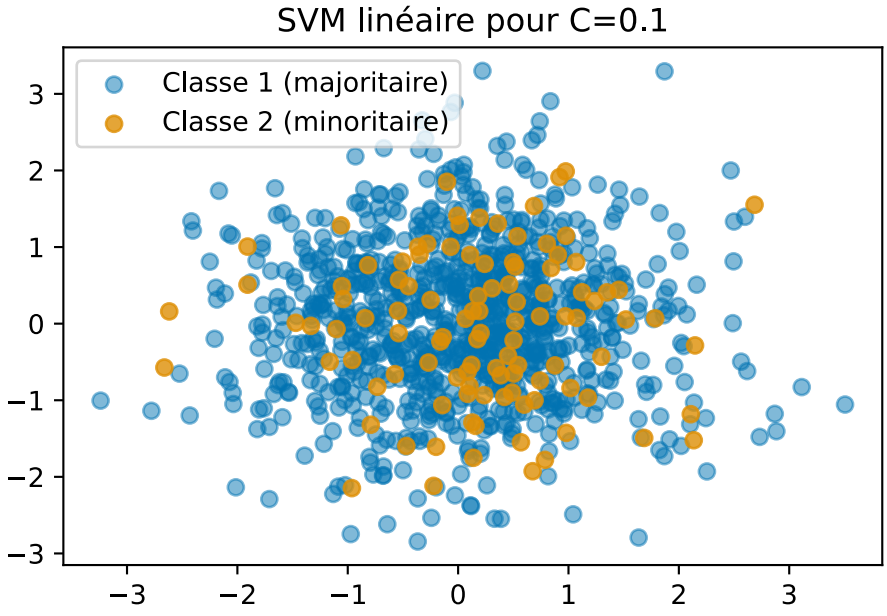
Commençons par générer le jeu de données très déséquilibré :

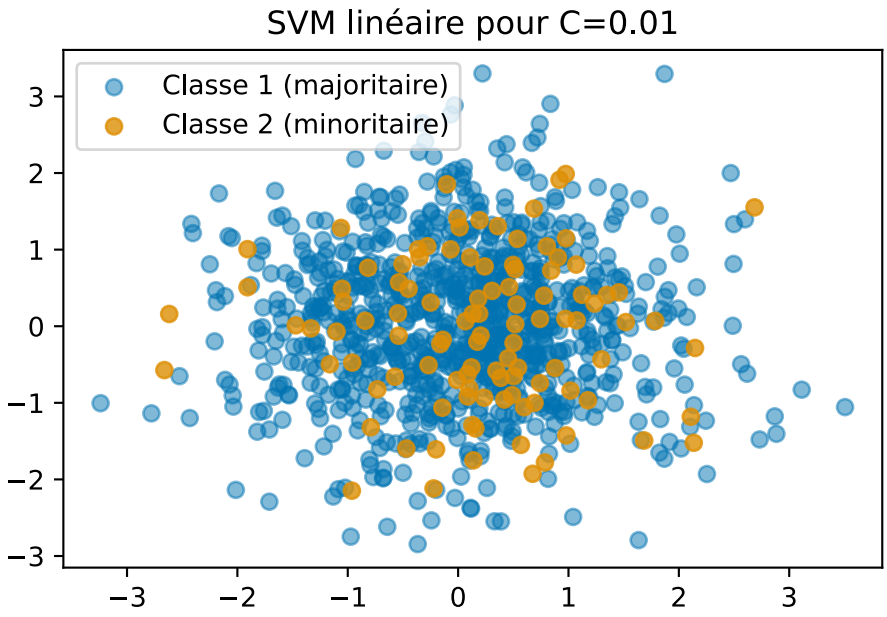
```
import sys
sys.path.append("scripts_python") # pour aller chercher la fonction de svm_source

import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from svm_source import *
# Création du jeu de données

n = 1000 # nombre d'observations
classes = np.random.choice([0,1], size=n, p=[0.9,0.1]) # classes 1 et 2 avec
respectivement p=90% et p=10%

# Affichage pour vérifier
print(np.sort(classes))

# Variables pour SVM
X = np.random.randn(n, 2)
Y = classes

def plot_svm(C_value):
    clf = SVC(kernel='linear', C=C_value)
    clf.fit(X, Y)

    plt.figure()
    plt.title(f"SVM linéaire pour C={C_value}")

    # Tracer les points
   plt.scatter(X[Y == 0][:, 0], X[Y == 0][:, 1], label="Classe 1 (majoritaire)",
alpha=0.5)
   plt.scatter(X[Y == 1][:, 0], X[Y == 1][:, 1], label="Classe 2 (minoritaire)",
alpha=0.8)

    # Tracer la frontière
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx, yy = np.meshgrid(
        np.linspace(xlim[0], xlim[1], 100),
        np.linspace(ylim[0], ylim[1], 100)
    )
```

```
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contour(xx, yy, Z, levels=[0], linewidths=2, linestyles="--", alpha=0.7)
    plt.legend()
    plt.show()

# Afficher plusieurs valeurs de C
for C in [1, 0.1, 0.01]:
    plot_svm(C)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1]
```


SVM linéaire pour C=1
Classe 1 (majoritaire)
Classe 2 (minoritaire)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


SVM linéaire pour C=0.1
Classe 1 (majoritaire)
Classe 2 (minoritaire)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

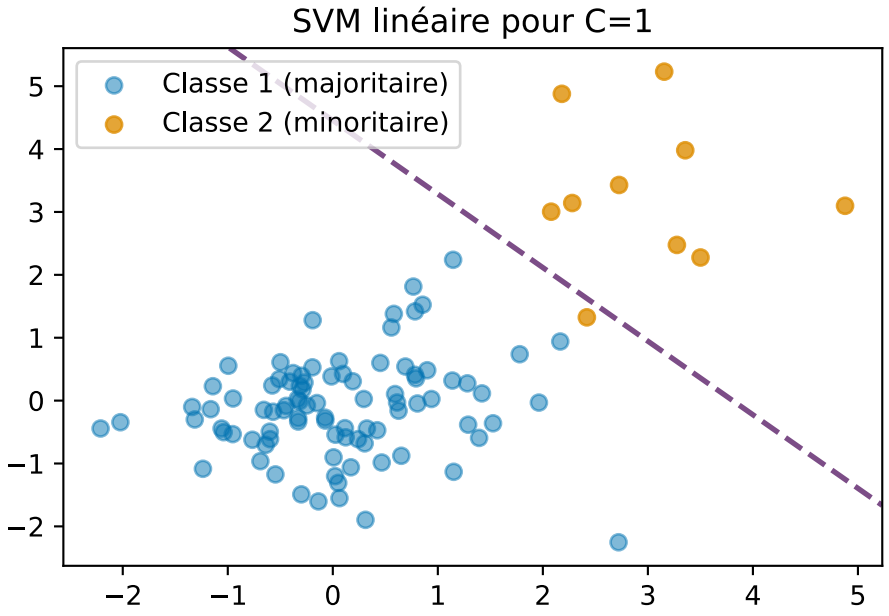


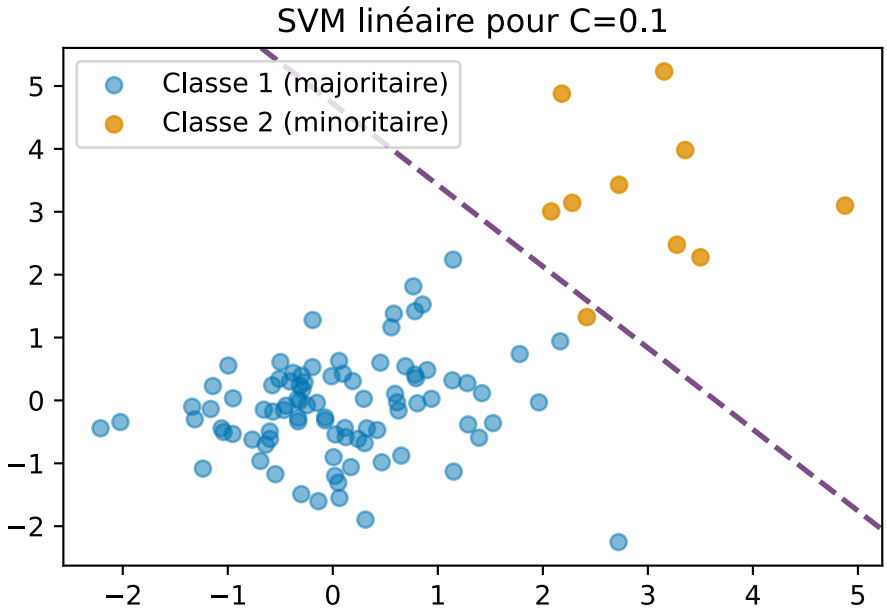
Sur un jeu de données purement aléatoire, on observe que le paramètre C n'a aucune influence, les frontières n'apparaissent même pas. Ceci est logique car les données sont impossibles à séparer facilement, il n'y a aucune logique sous-jacente à cette répartition des points.

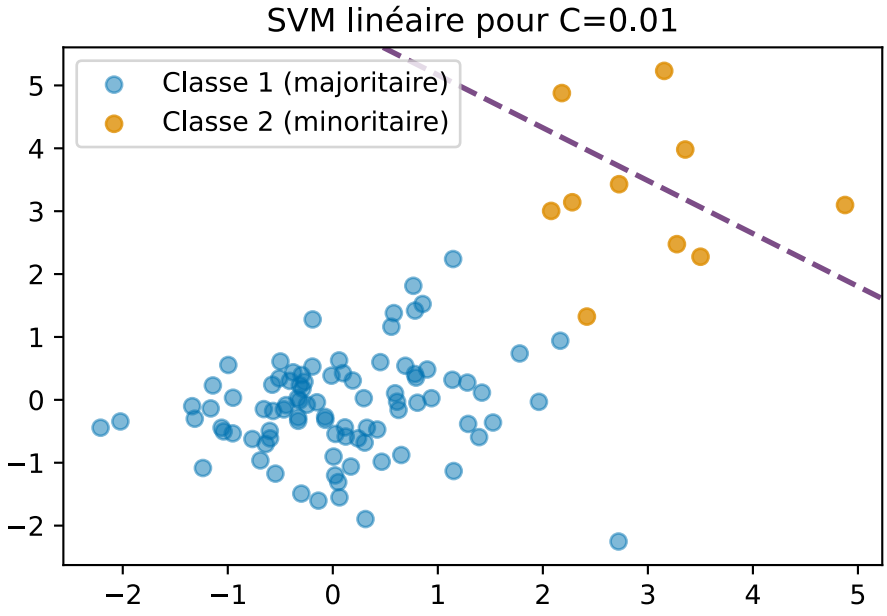
Si on prend maintenant un jeu de données plus structuré :

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

# Génération d'un dataset structuré (mais déséquilibré)
n_majoritaire = 90
n_minoritaire = 10

# Classe majoritaire centrée en (0,0)
X_majoritaire = np.random.randn(n_majoritaire, 2) * 0.8 + np.array([0, 0])
# Classe minoritaire centrée en (3,3)
X_minoritaire = np.random.randn(n_minoritaire, 2) * 0.8 + np.array([3, 3])

X = np.vstack((X_majoritaire, X_minoritaire))
Y = np.array([0]*n_majoritaire + [1]*n_minoritaire)

def plot_svm(C_value):
    clf = SVC(kernel='linear', C=C_value)
    clf.fit(X, Y)

    plt.figure()
    plt.title(f"SVM linéaire pour C={C_value}")
```

```
    # Tracer les points
    plt.scatter(X[Y == 0][:, 0], X[Y == 0][:, 1], label="Classe 1 (majoritaire)",
alpha=0.5)
    plt.scatter(X[Y == 1][:, 0], X[Y == 1][:, 1], label="Classe 2 (minoritaire)",
alpha=0.8)

    # Tracer la frontière
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx, yy = np.meshgrid(
        np.linspace(xlim[0], xlim[1], 100),
        np.linspace(ylim[0], ylim[1], 100)
    )
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contour(xx, yy, Z, levels=[0], linewidths=2, linestyles="--", alpha=0.7)
    plt.legend()
    plt.show()

# Afficher plusieurs valeurs de C
for C in [1, 0.1, 0.01]:
    plot_svm(C)
```







Ici, on voit l'action du paramètre C, la frontière se déplace dans les différents exemples.

Question 4

```
#%%
##############################################################################
#               Face Recognition Task
##############################################################################
```

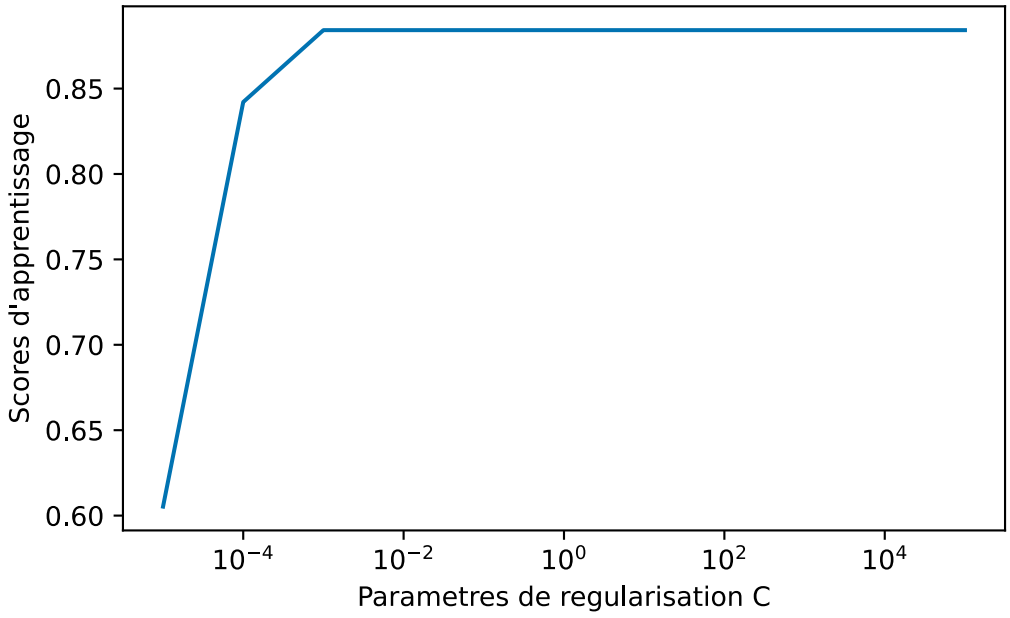
```
"""
The dataset used in this example is a preprocessed excerpt
of the "Labeled Faces in the Wild", aka LFW_:

  http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz (233MB)

  _LFW: http://vis-www.cs.umass.edu/lfw/
"""

########################################################################
# Download the data and unzip; then load it as numpy arrays
from sklearn.datasets import fetch_lfw_people

lfw_people = fetch_lfw_people(
    min_faces_per_person=70,
    resize=0.4,
    color=True,
    funneled=False,
    slice_=None,
    download_if_missing=True
)

lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4,
                              color=True, funneled=False, slice_=None,
                              download_if_missing=True)
# data_home='.'

# 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']
# names = ['Donald Rumsfeld', '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)

#%%
###############################################################################
# Split data into a half training and half test set
X_train, X_test, y_train, y_test, images_train, images_test = \
    train_test_split(X, y, images, test_size=0.5, random_state=0)
# X_train, X_test, y_train, y_test = \
#    train_test_split(X, y, test_size=0.5, random_state=0)

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]
images_train, images_test = images[
    train_idx, :, :, :], images[test_idx, :, :, :]

###############################################################################
# Quantitative evaluation of the model quality on the test set
```


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

# 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, random_state=42)
    clf.fit(X_train, y_train)
    # on mesure le score sur le jeu de test
    scores.append(clf.score(X_test, y_test))
    print("C=%f, score=%f" % (C, scores[-1]))
print("done in %0.3fs" % (time() - t0))
y_pred = clf.predict(X_test)
```

```
# get the best C


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
C=0.000010, score=0.605263
C=0.000100, score=0.842105
C=0.001000, score=0.884211
C=0.010000, score=0.884211
C=0.100000, score=0.884211
C=1.000000, score=0.884211
C=10.000000, score=0.884211
C=100.000000, score=0.884211
C=1000.000000, score=0.884211
C=10000.000000, score=0.884211
C=100000.000000, score=0.884211
done in 2.223s
Best C: 0.001
```




```
Best score: 0.8842105263157894
Predicting the people names on the testing set
```

Le paramètre de régularisation (C) influence directement la performance du SVM.
Pour des valeurs très faibles ((C = 10^{-5})), le modèle est trop régularisé et le score de prédiction reste faible (environs 0,60).
Lorsque (C) augmente, le score progresse rapidement et atteint un maximum autour de (C = 10^{-3}) (score ≈ 0,90).
Au-delà, le score se stabilise et n'apporte plus de gain. Le meilleur compromis biais/variance est donc obtenu pour (C = 0,001).

Question 5

```
def run_svm_cv(_X, _y):
    _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]

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

    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, 50, )
#with gaussian coefficients of std sigma
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
Generalization score for linear kernel: 1.0, 0.9157894736842105

Score avec variable de nuisance
Generalization score for linear kernel: 0.9947368421052631, 0.5842105263157895
```

L'ajout de variables de nuisance dégrade nettement la performance du SVM. En effet, si le score d'entraînement reste parfait (sur-apprentissage), le score de généralisation chute fortement (de 0.89 à 0.58 dans notre expérience).

Question 7

Le biais introduit par le code se situe dans ces deux lignes :

```
# Standardisation des données
X -= np.mean(X, axis=0) # on soustrait la moyenne
X /= np.std(X, axis=0) # on divise par l'écart type
```

En effet, ici on standardise les données avant de séparer notre jeu de données en 2 pour le train et le test, ce qui fait que l'on utilise à la fois les données du train et du test pour calculer la moyenne et l'écart-type qui vont ensuite être appliqués à l'ensemble de nos données pour les standardiser. Ceci crée un biais car les données de test ne doivent pas servir pour créer le modèle.

Conclusion

Ce TP nous a permis de nous familiariser avec le package scikit-learn de Python en l'utilisant pour effectuer des analyses sur différents types de jeux de données. L'influence du paramètre C a été étudiée en profondeur, revélant son importance dans le traçage final car il détermine le compromis

entre la capacité du modèle à classifier correctement la partie d'apprentissage et la maximisation de la marge.