



**STATISTIQUE**  
**SCIENCE DES DONNÉES**  
UNIVERSITÉ DE MONTPELLIER



---

# TP APPRENTISSAGE STATISTIQUE

---

Compte rendu

**Kilian Saint-Chely**

Professeur : Bilel Bensaid

Master Statistiques et Science des Données

Université de Montpellier

3 octobre 2025



# Table des matières

<b>1</b>	<b>Classification des iris</b>	<b>5</b>
1.1	Importation des librairies et préparation des données . . . . .	5
1.2	Classification à partir d'un noyau linéaire . . . . .	6
1.3	Classification à partir d'un noyau polynomial . . . . .	6
1.4	Visualisation de nos classifications . . . . .	6
1.5	Reprise d'un noyau polynomial en retirant le degré 1 . . . . .	7
<b>2</b>	<b>Classification des visages</b>	<b>9</b>
2.1	Influence du paramètre de régularisation . . . . .	9
2.2	Évaluation qualitative du modèle de prédiction et visualisation des coefficients	12
2.3	Influence de l'ajout de variables de nuisance . . . . .	13
2.4	Amélioration de la prédiction à l'aide d'une réduction de dimension . . . . .	14
2.5	Biais dans le prétraitement des données . . . . .	14
<b>3</b>	<b>Code complet</b>	<b>15</b>



# Chapitre 1

## Classification des iris

### 1.1 Importation des librairies et préparation des données

Avant de commencer, nous importons les librairies et le jeu de données nécessaire au TP

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.svm import SVC
4
5 from svm_source import *
6 from sklearn import svm
7 from sklearn import datasets
8 from sklearn.utils import shuffle
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import train_test_split, GridSearchCV
11 from sklearn.datasets import fetch_lfw_people
12 from sklearn.decomposition import PCA
13 from time import time
14
15 scaler = StandardScaler()
16
17 import warnings
18 warnings.filterwarnings("ignore")
19
20 plt.style.use('ggplot')
21
22 iris = datasets.load_iris()
23 X = iris.data
24 X = scaler.fit_transform(X)
25 y = iris.target
26 X = X[y != 0, :2]
27 y = y[y != 0]
```

Listing 1.1 – Préparation

Ensuite nous séparons les données dans l'optique de réaliser une validation croisée. On garde 25% de nos données pour le test et 75% pour l'apprentissage. Enfin, nous fixons une graine pour la reproductibilité du TP.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.25, random_state=42, shuffle=True)
```

Listing 1.2 – Séparation pour validation croisée

## 1.2 Classification à partir d'un noyau linéaire

Dans cette section, nous mettons en place un code qui va classer la classe 1 contre la classe 2 à l'aide d'un noyau linéaire. Nous regardons alors le meilleur paramètre  $C$  et les scores sur les données d'entraînement et de tests.

```

1 # fit the model and select the best hyperparameter C
2 parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3, 200))
3               }
4 clf_linear = GridSearchCV(SVC(), parameters, n_jobs=-1)
5 clf_linear.fit(X_train, y_train)
6
7 # compute the score
8 print(clf_linear.best_params_)
9 print('Generalization score for linear kernel: %s, %s' %
10       (clf_linear.score(X_train, y_train),
11        clf_linear.score(X_test, y_test)))

```

Listing 1.3 – Classification des classes 1 contre 2 avec noyau linéaire

Les observations que l'on peut faire sont les suivantes :

- le meilleur paramètre est  $C=0.84$
- le score du jeu d'entraînement est 0.75
- le score sur l'ensemble test est 0.68 ( $<0.75$ , ce qui est logique puisqu'un modèle colle toujours mieux aux données d'entraînement)

## 1.3 Classification à partir d'un noyau polynomial

Nous effectuons à présent la même classification mais avec un noyau polynomial où on teste les degrés 1, 2 et 3.

```

1 Cs = list(np.logspace(-3, 3, 5))
2 gammas = 10. ** np.arange(1, 2)
3 degrees = np.r_[1, 2, 3]
4
5 # fit the model and select the best set of hyperparameters
6 parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree':
7               degrees}
8
9 clf_poly = GridSearchCV(SVC(), parameters, n_jobs=-1)
10 clf_poly.fit(X_train, y_train)
11
12 print(clf_poly.best_params_)
13 print('Generalization score for polynomial kernel: %s, %s' %
14       (clf_poly.score(X_train, y_train),
15        clf_poly.score(X_test, y_test)))

```

Listing 1.4 – Classification des classes 1 contre 2 avec noyau polynomial

Il ressort de ce test, que le noyau polynomial le plus performant est celui de degré 1 ; avec exactement le même score que le noyau linéaire. Ceci est cohérent puisque le degré d'un noyau linéaire est 1.

## 1.4 Visualisation de nos classifications

On écrit un code permettant de visualiser les données et les frontières que nous avons tracées.

```

1 # display your results using frontiere (svm_source.py)
2
3 def f_linear(xx):
4     return clf_linear.predict(xx.reshape(1, -1))
5
6 def f_poly(xx):
7     return clf_poly.predict(xx.reshape(1, -1))
8
9 plt.ion()
10 plt.figure(figsize=(15, 5))
11 plt.subplot(131)
12 plot_2d(X, y)
13 plt.title("iris_dataset")
14
15 plt.subplot(132)
16 frontiere(f_linear, X, y)
17 plt.title("linear_kernel")
18
19 plt.subplot(133)
20 frontiere(f_poly, X, y)
21 plt.title("polynomial_kernel")
22
23 plt.tight_layout()
24 plt.draw()

```

Listing 1.5 – Visualisation des données et des classifications

On peut constater sur la figure 1.1 que la frontière est sensiblement différente malgré un score de performance identique. Cela s'explique par le fait que le noyau linéaire est sans terme de biais, alors que le noyau polynomial de degré 1 est une forme affine (avec biais).

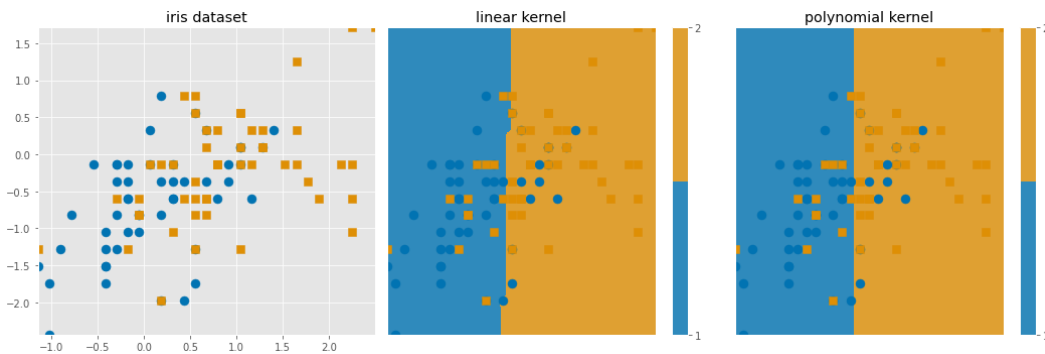


FIGURE 1.1 – Visualisation des données et des frontières

## 1.5 Reprise d'un noyau polynomial en retirant le degré 1

Dans cette partie, nous reprenons le même procédé que les deux sections précédentes, mais en retirant la possibilité de degré 1 pour le noyau polynomial afin de visualiser ce que cela donnerait comme résultat.

```

1 Cs = list(np.logspace(-3, 3, 5))
2 gammas = 10. ** np.arange(1, 2)
3 degrees = np.r_[2, 3]
4

```

```

5 # fit the model and select the best set of hyperparameters
6 parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree':
   degrees}
7
8 clf_poly2 = GridSearchCV(SVC(), parameters, n_jobs=-1)
9 clf_poly2.fit(X_train, y_train)
10
11 print(clf_poly2.best_params_)
12 print('Generalization score for polynomial kernel: %s, %s' %
13       (clf_poly2.score(X_train, y_train),
14        clf_poly2.score(X_test, y_test)))
15
16 def f_poly2(xx):
17     return clf_poly2.predict(xx.reshape(1, -1))
18
19 plt.ion()
20 plt.figure(figsize=(15, 5))
21 plt.subplot(131)
22 plot_2d(X, y)
23 plt.title("iris dataset")
24
25 plt.subplot(132)
26 frontiere(f_linear, X, y)
27 plt.title("linear kernel")
28
29 plt.subplot(133)
30 frontiere(f_poly2, X, y)
31 plt.title("polynomial kernel")
32
33 plt.tight_layout()
34 plt.draw()

```

Listing 1.6 – Programme de classification et de visualisation avec un noyau polynomial de degré supérieur à 1

Les résultats montrent que le noyau polynomial le plus approprié est celui de degré 2. Les scores sont :

- jeu d'entraînement : 0.65
- ensemble test : 0.52

On peut voir sur la figure 1.2 que la frontière est bien différente de celle tracée à partir d'un noyau linéaire.

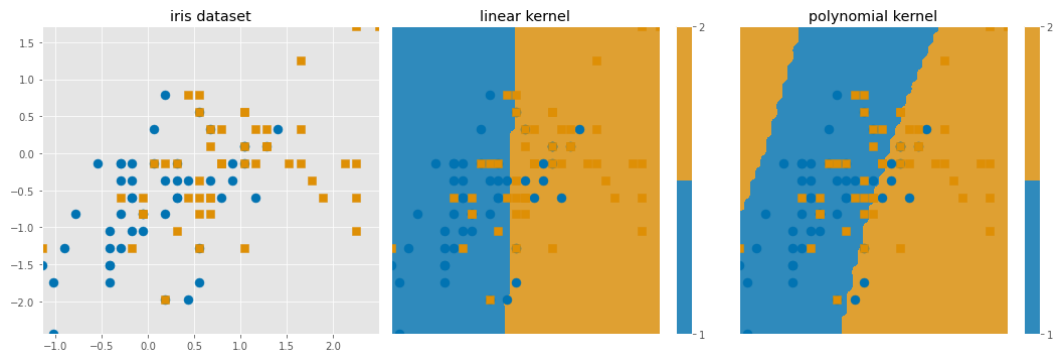


FIGURE 1.2 – Visualisation des données et des frontières avec noyau polynomial de degré 2



## Chapitre 2

# Classification des visages

### 2.1 Influence du paramètre de régularisation

Dans un premier temps, nous importons, préparons et nous familiarisons avec les données.

```
1 """
2 The dataset used in this example is a preprocessed excerpt
3 of the "Labeled Faces in the Wild", aka LFW:
4
5 http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz (233MB)
6
7 LFW: http://vis-www.cs.umass.edu/lfw/
8 """
9
10 # Download the data and unzip; then load it as numpy arrays
11 lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4,
12                               color=True, funneled=False, slice_=None,
13                               download_if_missing=True)
14
15 # introspect the images arrays to find the shapes (for plotting)
16 images = lfw_people.images
17 n_samples, h, w, n_colors = images.shape
18
19 # the label to predict is the id of the person
20 target_names = lfw_people.target_names.tolist()
21
22 # Pick a pair to classify
23 names = ['Tony Blair', 'Colin Powell']
24
25 idx0 = (lfw_people.target == target_names.index(names[0]))
26 idx1 = (lfw_people.target == target_names.index(names[1]))
27 images = np.r_[images[idx0], images[idx1]]
28 n_samples = images.shape[0]
29 y = np.r_[np.zeros(np.sum(idx0)), np.ones(np.sum(idx1))].astype(int)
30
31 # plot a sample set of the data
32 plot_gallery(images, np.arange(12))
33 plt.show()
34
35 # Extract features (grayscale, mean per pixel)
36 X = (np.mean(images, axis=3)).reshape(n_samples, -1)
37
38 # Scale features
39 X -= np.mean(X, axis=0)
```

```

40 X /= np.std(X, axis=0)
41
42 # Split data into a half training and half test set
43 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)

```

Listing 2.1 – Mise en place des données

Nous regardons à présent l'influence de C en le faisant varier.

```

1  print("---Linear kernel---")
2  print("Fitting the classifier to the training set")
3  t0 = time()
4
5  Cs = 10. ** np.arange(-5, 6) # from 1e-5 to 1e5
6  scores = []
7  for C in Cs:
8      clf = SVC(kernel='linear', C=C)
9      clf.fit(X_train, y_train)
10     scores.append(clf.score(X_test, y_test))
11
12 ind = np.argmax(scores)
13 print("Best C: {}".format(Cs[ind]))
14
15 plt.figure()
16 plt.plot(Cs, scores, marker="o")
17 plt.xlabel("Paramètre de régularisation C")
18 plt.ylabel("Score de test")
19 plt.xscale("log")
20 plt.title("Influence de C")
21 plt.tight_layout()
22 plt.show()
23 print("Best score: {}".format(np.max(scores)))

```

Listing 2.2 – Programme de visualisation de l'influence de C

Ce code nous dit que la meilleure valeur de C est 0.001 et que le score associé est 0.88. Nous pouvons visualiser ce résultat sur la figure 2.1

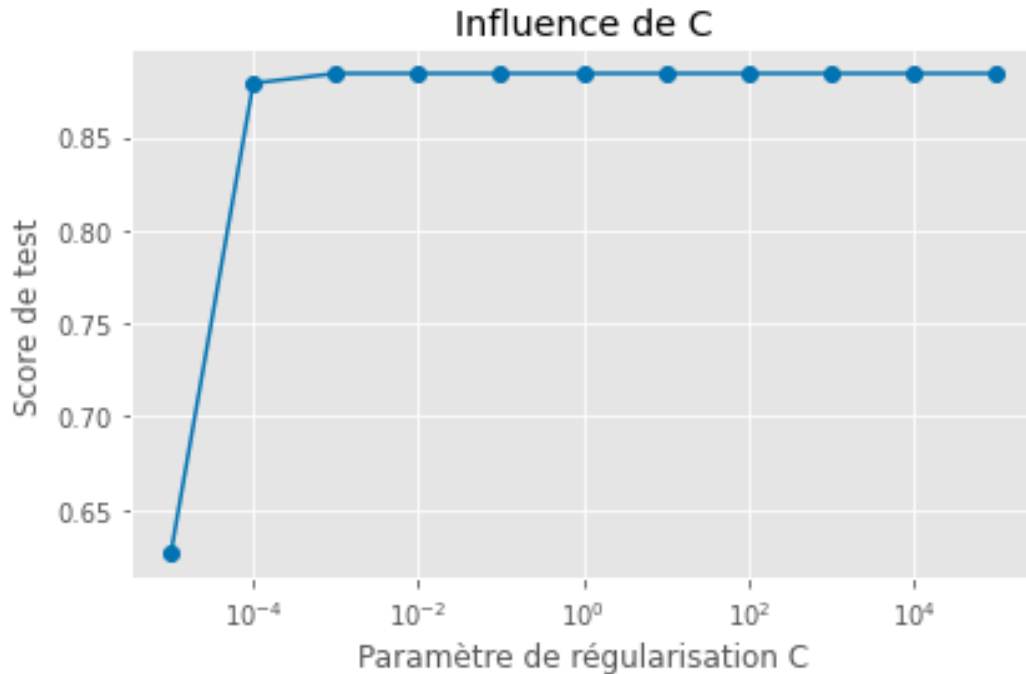


FIGURE 2.1 – Score de test en fonction de la valeur du paramètre de régularisation C

A présent, nous réentraînon le modèle avec la meilleure valeur de C trouvée, nous effectuons une prédiction sur les données de test. Nous prenons soin de mesurer le temps d'exécution et nous comparons la précision réelle du modèle avec le niveau de hasard.

```

1 print("Predicting the people names on the testing set")
2 t0 = time()
3
4 # retrain best model
5 clf = SVC(kernel='linear', C=Cs[ind])
6 clf.fit(X_train, y_train)
7 y_pred = clf.predict(X_test)
8
9 print("done in %0.3fs" % (time() - t0))
10 # The chance level is the accuracy that will be reached when
    constantly predicting the majority class.
11 print("Chance level: %s" % max(np.mean(y), 1. - np.mean(y)))
12 print("Accuracy: %s" % clf.score(X_test, y_test))

```

Listing 2.3 – Utilisation de la meilleure valeur de C

Les résultats obtenus sont les suivants :

- temps d'exécution : 1.149 secondes
- niveau de hasard : 0.62
- précision réelle du modèle : 0.88

On constate sans surprise que le modèle est "utile" puisqu'il est plus performant que le hasard.

## 2.2 Évaluation qualitative du modèle de prédiction et visualisation des coefficients

Nous voulons à présent faire un test qualitatif sur la prédiction d'image et visualiser les coefficients appris par le modèle.

```

1 # Qualitative evaluation of the predictions using matplotlib
2
3 prediction_titles = [title(y_pred[i], y_test[i], names)
4                       for i in range(y_pred.shape[0])]
5
6 plot_gallery(images_test, prediction_titles)
7 plt.show()
8
9 # Look at the coefficients
10 plt.figure()
11 plt.imshow(np.reshape(clf.coef_, (h, w)))
12 plt.show()

```

Listing 2.4 – Test de prédiction et visualisation des coefficients

On obtient dans ce cas là une prédiction 100% juste malgré un score théorique de 88% (figure 2.2). On peut voir sur la figure 2.3 les coefficients, c'est-à-dire les traits discriminants pour le modèle, qui lui permettent de prédire une image.



FIGURE 2.2 – Prédiction de 12 visages

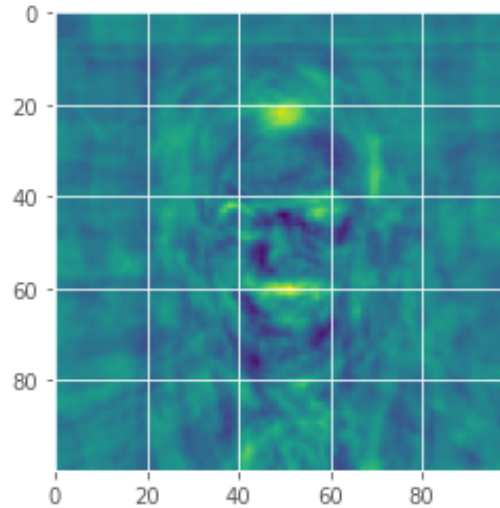


FIGURE 2.3 – Visualisation des coefficients appris par le modèle

## 2.3 Influence de l'ajout de variables de nuisance

Nous voulons tester l'influence de l'ajout de variables de nuisance. Pour cela, nous calculons le score sans variables de nuisance puis nous ajoutons 300 variables de nuisance Gaussiennes réduites.

```

1 def run_svm_cv(_X, _y):
2     _indices = np.random.permutation(_X.shape[0])
3     _train_idx, _test_idx = _indices[:_X.shape[0] // 2], _indices[_X.
4         shape[0] // 2:]
5     _X_train, _X_test = _X[_train_idx, :], _X[_test_idx, :]
6     _y_train, _y_test = _y[_train_idx], _y[_test_idx]
7
8     _parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3,
9         5)))}
10    _svr = svm.SVC()
11    _clf_linear = GridSearchCV(_svr, _parameters)
12    _clf_linear.fit(_X_train, _y_train)
13
14    print('Generalization score for linear kernel: %s, %s\n' %
15        (_clf_linear.score(_X_train, _y_train), _clf_linear.score(
16            _X_test, _y_test)))
17
18    print("Score sans variable de nuisance")
19    run_svm_cv(X, y)
20
21    print("Score avec variable de nuisance")
22    n_features = X.shape[1]
23    # On rajoute des variables de nuisance
24    sigma = 1
25    noise = sigma * np.random.randn(n_samples, 300, )
26    #with gaussian coefficients of std sigma
27    X_noisy = np.concatenate((X, noise), axis=1)
28    X_noisy = X_noisy[np.random.permutation(X.shape[0])]
29    run_svm_cv(X_noisy, y)

```

Listing 2.5 – Programme de calcul de l'influence de l'ajout de variables de nuisance

En généralisation, nous avons un score sans variable de nuisance de 0.94 contre un score avec variables de nuisance de 0.58. On peut donc en conclure que l'ajout de variables de nuisance fait chuter la performance du modèle.

## 2.4 Amélioration de la prédiction à l'aide d'une réduction de dimension

Nous effectuons une ACP afin de réduire la dimension et de voir si cela améliore comme on peut le supposer la prédiction.

```
1 print("Score après réduction de dimension")
2
3 n_components = 80 # jouer avec ce parametre
4 pca = PCA(n_components=n_components, svd_solver='randomized').fit(
5     X_noisy)
6 X_pca = pca.transform(X_noisy)
7 run_svm_cv(X_pca, y)
```

Listing 2.6 – Prédiction avec l'utilisation de l'ACP pour réduire la dimension

Nous obtenons un score de 0.83 sur l'échantillon d'entraînement et de 0.52 sur l'échantillon test. Ce qui paraît anormal car une sélection de variables sachant que 300 d'entre elles sont du bruit (nous le savons puisque c'est nous qui les avons ajoutées) devrait augmenter le score de prédiction.

## 2.5 Biais dans le prétraitement des données

En prenant du recul sur notre travail, on peut constater qu'il y a un biais dans le prétraitement de nos données. En effet, nous effectuons la moyenne sur les canaux de couleurs, pour ne garder que l'illumination moyenne par pixel, ce qui engendre une perte d'information.

```
1 X = (np.mean(images, axis=3)).reshape(n_samples, -1)
```

Listing 2.7 – Hypothèse d'une commande source de perte d'information

## Chapitre 3

# Code complet

```
1  #%%
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from sklearn.svm import SVC
5
6  from svm_source import *
7  from sklearn import svm
8  from sklearn import datasets
9  from sklearn.utils import shuffle
10 from sklearn.preprocessing import StandardScaler
11 from sklearn.model_selection import train_test_split, GridSearchCV
12 from sklearn.datasets import fetch_lfw_people
13 from sklearn.decomposition import PCA
14 from time import time
15
16 scaler = StandardScaler()
17
18 import warnings
19 warnings.filterwarnings("ignore")
20
21 plt.style.use('ggplot')
22
23 #%%
24 #####
25 # Toy dataset : 2 gaussians
26 #####
27
28 n1 = 200
29 n2 = 200
30 mu1 = [1., 1.]
31 mu2 = [-1./2, -1./2]
32 sigma1 = [0.9, 0.9]
33 sigma2 = [0.9, 0.9]
34 X1, y1 = rand_bi_gauss(n1, n2, mu1, mu2, sigma1, sigma2)
35
36 plt.show()
37 plt.close("all")
38 plt.ion()
39 plt.figure(1, figsize=(15, 5))
40 plt.title('First data set')
41 plot_2d(X1, y1)
42
```

```

43 X_train = X1[:, :2]
44 Y_train = y1[:, :2].astype(int)
45 X_test = X1[1:, :2]
46 Y_test = y1[1:, :2].astype(int)
47
48 # fit the model with linear kernel
49 clf = SVC(kernel='linear')
50 clf.fit(X_train, Y_train)
51
52 # predict labels for the test data base
53 y_pred = clf.predict(X_test)
54
55 # check your score
56 score = clf.score(X_test, Y_test)
57 print('Score: %s' % score)
58
59 # display the frontiere
60 def f(xx):
61     """Classifier: needed to avoid warning due to shape issues"""
62     return clf.predict(xx.reshape(1, -1))
63
64 plt.figure()
65 frontiere(f, X_train, Y_train, w=None, step=50, alpha_choice=1)
66
67 # Same procedure but with a grid search
68 parameters = {'kernel': ['linear'], 'C': list(np.linspace(0.001, 3,
69     21))}
70 clf2 = SVC()
71 clf_grid = GridSearchCV(clf2, parameters, n_jobs=-1)
72 clf_grid.fit(X_train, Y_train)
73
74 # check your score
75 print(clf_grid.best_params_)
76 print('Score: %s' % clf_grid.score(X_test, Y_test))
77
78 def f_grid(xx):
79     """Classifier: needed to avoid warning due to shape issues"""
80     return clf_grid.predict(xx.reshape(1, -1))
81
82 # display the frontiere
83 plt.figure()
84 frontiere(f_grid, X_train, Y_train, w=None, step=50, alpha_choice=1)
85
86 """
87 # Iris Dataset
88 """
89
90 iris = datasets.load_iris()
91 X = iris.data
92 X = scaler.fit_transform(X)
93 y = iris.target
94 X = X[y != 0, :2]
95 y = y[y != 0]
96
97 # split train test (say 25% for the test)
98 # Split train/test
99 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.25, random_state=42, shuffle=True)

```



```

100
101 #####
102 # fit the model with linear vs polynomial kernel
103 #####
104
105 #%%
106 # Q1 Linear kernel
107
108 # fit the model and select the best hyperparameter C
109 parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3, 200))
110              }
111 clf_linear = GridSearchCV(SVC(), parameters, n_jobs=-1)
112 clf_linear.fit(X_train, y_train)
113
114 # compute the score
115 print(clf_linear.best_params_)
116 print('Generalization score for linear kernel: %s, %s' %
117       (clf_linear.score(X_train, y_train),
118        clf_linear.score(X_test, y_test)))
119
120 #%%
121 # Q2 polynomial kernel
122 Cs = list(np.logspace(-3, 3, 5))
123 gammas = 10. ** np.arange(1, 2)
124 degrees = np.r_[1, 2, 3]
125
126 # fit the model and select the best set of hyperparameters
127 parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree':
128              degrees}
129
130
131 clf_poly = GridSearchCV(SVC(), parameters, n_jobs=-1)
132 clf_poly.fit(X_train, y_train)
133
134 print(clf_poly.best_params_)
135 print('Generalization score for polynomial kernel: %s, %s' %
136       (clf_poly.score(X_train, y_train),
137        clf_poly.score(X_test, y_test)))
138
139 #sans le degre 1
140 Cs = list(np.logspace(-3, 3, 5))
141 gammas = 10. ** np.arange(1, 2)
142 degrees = np.r_[2, 3]
143
144 # fit the model and select the best set of hyperparameters
145 parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree':
146              degrees}
147
148
149 clf_poly2 = GridSearchCV(SVC(), parameters, n_jobs=-1)
150 clf_poly2.fit(X_train, y_train)
151
152 print(clf_poly2.best_params_)
153 print('Generalization score for polynomial kernel: %s, %s' %
154       (clf_poly2.score(X_train, y_train),
155        clf_poly2.score(X_test, y_test)))
156
157 #%%
158 # display your results using frontiere (svm_source.py)
159

```

```

156 def f_linear(xx):
157     return clf_linear.predict(xx.reshape(1, -1))
158
159 def f_poly(xx):
160     return clf_poly.predict(xx.reshape(1, -1))
161
162 plt.ion()
163 plt.figure(figsize=(15, 5))
164 plt.subplot(131)
165 plot_2d(X, y)
166 plt.title("iris_dataset")
167
168 plt.subplot(132)
169 frontiere(f_linear, X, y)
170 plt.title("linear_kernel")
171
172 plt.subplot(133)
173 frontiere(f_poly, X, y)
174 plt.title("polynomial_kernel")
175
176 plt.tight_layout()
177 plt.draw()
178
179 def f_poly2(xx):
180     return clf_poly2.predict(xx.reshape(1, -1))
181
182 plt.ion()
183 plt.figure(figsize=(15, 5))
184 plt.subplot(131)
185 plot_2d(X, y)
186 plt.title("iris_dataset")
187
188 plt.subplot(132)
189 frontiere(f_linear, X, y)
190 plt.title("linear_kernel")
191
192 plt.subplot(133)
193 frontiere(f_poly2, X, y)
194 plt.title("polynomial_kernel")
195
196 plt.tight_layout()
197 plt.draw()
198
199 #%%
200 #####
201 #                               Face Recognition Task
202 #####
203 """
204 The dataset used in this example is a preprocessed excerpt
205 of the "Labeled Faces in the Wild", aka LFW:
206
207 http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz (233MB)
208
209 LFW: http://vis-www.cs.umass.edu/lfw/
210 """
211
212 #####
213 # Download the data and unzip; then load it as numpy arrays
214 lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4,

```

```

215         color=True, funneled=False, slice_=None,
216         download_if_missing=True)
217
218 # introspect the images arrays to find the shapes (for plotting)
219 images = lfw_people.images
220 n_samples, h, w, n_colors = images.shape
221
222 # the label to predict is the id of the person
223 target_names = lfw_people.target_names.tolist()
224
225 #####
226 # Pick a pair to classify
227 names = ['Tony_Blair', 'Colin_Powell']
228
229 idx0 = (lfw_people.target == target_names.index(names[0]))
230 idx1 = (lfw_people.target == target_names.index(names[1]))
231 images = np.r_[images[idx0], images[idx1]]
232 n_samples = images.shape[0]
233 y = np.r_[np.zeros(np.sum(idx0)), np.ones(np.sum(idx1))].astype(int)
234
235 # plot a sample set of the data
236 plot_gallery(images, np.arange(12))
237 plt.show()
238
239 #####
240 # Extract features (grayscale, mean per pixel)
241 X = (np.mean(images, axis=3)).reshape(n_samples, -1)
242
243 # Scale features
244 X -= np.mean(X, axis=0)
245 X /= np.std(X, axis=0)
246
247 #####
248 # Split data into a half training and half test set
249 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)
250
251 #####
252
253 # Q4 Influence of regularization parameter C
254 print("---Linear kernel---")
255 print("Fitting the classifier to the training set")
256 t0 = time()
257
258 Cs = 10. ** np.arange(-5, 6) # from 1e-5 to 1e5
259 scores = []
260 for C in Cs:
261     clf = SVC(kernel='linear', C=C)
262     clf.fit(X_train, y_train)
263     scores.append(clf.score(X_test, y_test))
264
265 ind = np.argmax(scores)
266 print("Best C: {}".format(Cs[ind]))
267
268 plt.figure()
269 plt.plot(Cs, scores, marker="o")
270 plt.xlabel("Parameter regularisation C")
271 plt.ylabel("Score de test")
272 plt.xscale("log")

```

```

273 plt.title("Influence de C")
274 plt.tight_layout()
275 plt.show()
276 print("Best score: {}".format(np.max(scores)))
277
278 print("Predicting the people names on the testing set")
279 t0 = time()
280
281 # retrain best model
282 clf = SVC(kernel='linear', C=Cs[ind])
283 clf.fit(X_train, y_train)
284 y_pred = clf.predict(X_test)
285
286 print("done in {} % (time() - t0))".format(int((time() - t0) * 100)))
287 # The chance level is the accuracy that will be reached when
    constantly predicting the majority class.
288 print("Chance level: {}".format(max(np.mean(y), 1. - np.mean(y))))
289 print("Accuracy: {}".format(clf.score(X_test, y_test)))
290
291 #####
292
293 #%%
294 #####
295 # Qualitative evaluation of the predictions using matplotlib
296
297 prediction_titles = [title(y_pred[i], y_test[i], names)
298                      for i in range(y_pred.shape[0])]
299
300 plot_gallery(images_test, prediction_titles)
301 plt.show()
302
303 #####
304 # Look at the coefficients
305 plt.figure()
306 plt.imshow(np.reshape(clf.coef_, (h, w)))
307 plt.show()
308
309 #%%
310 # Q5
311
312
313 def run_svm_cv(_X, _y):
314     _indices = np.random.permutation(_X.shape[0])
315     _train_idx, _test_idx = _indices[:_X.shape[0] // 2], _indices[_X.
        shape[0] // 2:]
316     _X_train, _X_test = _X[_train_idx, :], _X[_test_idx, :]
317     _y_train, _y_test = _y[_train_idx], _y[_test_idx]
318
319     _parameters = {'kernel': ['linear'], 'C': list(np.logspace(-3, 3,
        5))}
320     _svr = svm.SVC()
321     _clf_linear = GridSearchCV(_svr, _parameters)
322     _clf_linear.fit(_X_train, _y_train)
323
324     print('Generalization score for linear kernel: {} s, {} s\n' %
325           (_clf_linear.score(_X_train, _y_train), _clf_linear.score(
326             _X_test, _y_test)))
327
328     print("Score sans variable de nuisance")

```

```

328 run_svm_cv(X, y)
329
330 print("Score_avec_variable_de_nuisance")
331 n_features = X.shape[1]
332 # On rajoute des variables de nuisances
333 sigma = 1
334 noise = sigma * np.random.randn(n_samples, 300, )
335 #with gaussian coefficients of std sigma
336 X_noisy = np.concatenate((X, noise), axis=1)
337 X_noisy = X_noisy[np.random.permutation(X.shape[0])]
338 run_svm_cv(X_noisy, y)
339
340 #%%
341 # Q6
342 print("Score_apres_reduction_de_dimension")
343
344 n_components = 100 # jouer avec ce parametre
345 pca = PCA(n_components=n_components, svd_solver='randomized').fit(
    X_noisy)
346 X_pca = pca.transform(X_noisy)
347 run_svm_cv(X_pca, y)

```

Listing 3.1 – Code python complet