Machine Learning

Elève VASSON KEVIN
Enseignants KESSACI ROJSSE_

TP Machine Learning

_bjectif est d'apprendre des principales techniques d'apprentissage supervisé et de se familiariser avec la bibliothèque python Scikit-learn (http://scikit-learn.org/stable/index.html). Cette bibliothèque est destinée à l'apprentissage automatique (le machine learning en anglais). Nous nous focaliserons notre étude sur des problèmes et donc des algorithmes/techniques de classification.

Nous allons voir pour la classification des

- Arbres de décisions
- SVM
- Techniques de bagging et random forest
- Techniques de boosting (ADABoost et XGBoost)

différents jeux de données

- Iris
- digits
- breast cancer

et nous nous intéresserons aux techniques de répartition *train/test*, à la normalisation des données et au paramétrage automatique.

Séance 1

Pour cette séance, nous allons découvrir scikit-learn et nous allons tester plusieurs méthodologies

Chargement des jeux de données

Scikit-learn embarque plusieurs jeux de données 'jouets' qui permettent de découvrir toutes les fonctionnalités de la bibliothèque. Il suffit alors d'importer le jeu de données *jeuXXX* avec la commande `from sklearn.datasets import load_jeuXXX` puis de le charger dans une variable `jXXX` avec la commande `jXXX = load_jeuXXX()`

Dans la suite nous considérons que les données sont rangées dans l'objet X et que la classe associée à chaque observation de X sont dans y. Les méthodes `data` et `target` permet d'accéder aux informations pour instancier X et y respectivement : `X, y = jXXX.data, jXXX.target`

La description du jeu de données est accessible via la méthode `DESCR` et peut être affichée via un simple `print(jXXX.DESCR)`.

Il est possible d'importer les jeux de données directement sous un format dataframe Pandas en ajoutant l'option en paramètre `as_frame= True` dans la commande de chargement et d'utiliser la méthode `frame` pour y accéder. `jXXX.frame` correspond donc au dataframe du jeu de données et peut être affiché via un print.

§ Taper les commandes pour importer et charger les jeux de données iris, digits et breast_cancer

§ Donner pour chacun des jeux de données : une description brève, le nombre d observations, le nombre de features, le nombre de classes et la répartition pour chacune d elles

Vous pouvez utiliser le code suivant pour afficher les 10 premières images du jeu de données digits

```
import matplotlib.pyplot as plt fig = plt.figure() for i, digit in enumerate(digits.images[:10]):     fig.add_subplot(1,10,i+1) plt.imshow(digit) plt.show()
```

Importation des packages nécessaires

In [185]:
```
from sklearn.datasets import load_iris,load_digits,load_breast_cancer
from sklearn import datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

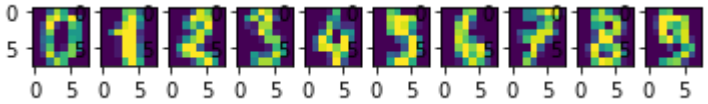
Importation des données

In [186]:
```
myiris, mydigits, mybreast_cancer = load_iris(as_frame= True), load_d
```

Table Digits

Affichage des 10 premières images

In [187]:
```
fig = plt.figure()
for i, digit in enumerate(mydigits.images[:10]):
    fig.add_subplot(1,10,i+1)
    plt.imshow(digit)
plt.show()
```



Dimensions du jeu de données

In [188]:
```
mydigits.data.shape
```

Out[188]:
```
(1797, 64)
```

In [189]:
```
mydigits.DESCR
```

Out[189]:
```
".. _digits_dataset:\n\nOptical recognition of handwritten digits d
ataset\n--------------------------------------------------\n\n**Dat
a Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Nu
mber of Attributes: 64\n    :Attribute Information: 8x8 image of in
teger pixels in the range 0..16.\n    :Missing Attribute Values: No
ne\n    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Dat
e: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-
written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Op
tical+Recognition+of+Handwritten+Digits\n\nThe data set contains im
ages of hand-written digits: 10 classes where\neach class refers to
a digit.\n\nPreprocessing programs made available by NIST were used
to extract\nnormalized bitmaps of handwritten digits from a preprin
ted form. From a\ntotal of 43 people, 30 contributed to the trainin
g set and different 13\nto the test set. 32x32 bitmaps are divided
into nonoverlapping blocks of\n4x4 and the number of on pixels are
counted in each block. This generates\nan input matrix of 8x8 where
each element is an integer in the range\n0..16. This reduces dimens
ionality and gives invariance to small\ndistortions.\n\nFor info on
NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. C
andela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and
C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR
5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Metho
ds of Combining Multiple Classifiers and Their\n    Applications to
Handwritten Digit Recognition, MSc Thesis, Institute of\n    Gradua
te Studies in Science and Engineering, Bogazici University.\n  - E.
Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  -
Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n
Linear dimensionalityreduction using relevance weighted LDA. School
of\n    Electrical and Electronic Engineering Nanyang Technological
University.\n    2005.\n  - Claudio Gentile. A New Approximate Maxi
mal Margin Classification\n    Algorithm. NIPS. 2000.\n"
```

Le nombre d'attributs est 64

Le nombre de classes est 3 ce qui correspond à un effet

Table Iris

Affichage des 10 premières lignes

In [190]:
```
myiris.frame.head(10)
```

Out[190]:

	sepal length(cm)	sepal width(cm)	petal length(cm)	petal width(cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

Description de la table Iris

La table Iris est composée de 4 attributs

- sepal length
- sepal width
- petal length
- petal width

Et contient aussi une colonne target composée de 3 classes Iris

In [191]:
```
list(myiris.target_names)
```

Out[191]:
```
['setosa', 'versicolor', 'virginica']
```

Dimension du jeu de données

In [192]:
```
np.shape(myiris.frame)
```

Out[192]:
```
(150, 5)
```

L'objectif de cette table est de déterminer à partir d'un modèle statistique l'appartenance d'une Iris à l'une des 3 classes suivant les valeurs de ses attributs La colonne Target permet d'entraîner notre modèle

Table breast_cancer

Affichage des 10 premières lignes

In [193]:
```python
mybreast_cancer.frame.head(10)
```

Out[193]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1
7	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2
8	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2
9	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2

Description de la table Breast_cancer

La table Breast_cancer est composée de 30 attributs

Elle contient aussi une colonne target composée de 2 classes

In [194]:
```python
list(mybreast_cancer.target_names)
```

Out[194]: ['malignant', 'benign']

Dimension du jeu de données

In [195]:
```python
np.shape(mybreast_cancer.frame)
```

Out[195]: (569, 31)

L'objectif de cette table est de déterminer à partir d'un modèle statistique si une personne est à risque ou non suivant la valeurs de ses différents attributs. La colonne Target permet d'entraîner notre modèle.

Normalisation du jeu de données

Prenons le jeu breast-cancer. Les 30 features sont de dimensions très différentes (voir les valeurs min et max de chacunes), ce qui peut entraîner pour certaines méthodes de classification une performance diminuée.

Considérons la méthode SVM accessible via import `from sklearn.svm import SVC` qui est le module SVM adapté à la prédiction de classe à partir des objets X et y présentés ci-dessus.

créons les ensembles d'exemples d'apprentissage et de *train* et de *test*

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=s)
```

où c'est la partie que vous pouvez choisir. Il est nécessaire d'importer ce module avec la commande `from sklearn.model_selection import .train_test_split`

Instancions le svm puis entraînons le sur les ensembles X_train et y_train afin de prédire la classe des observations de X_test et de comparer la accuracy (importer le module accuracy_score avec la commande `from sklearn.metrics import accuracy_score`). Le code suivant met en œuvre cette méthodologie

```
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print("Accuracy: %.2f%%" % (accuracy_score(y_test,y_pred) *
100.0))
```

§ Quelle valeur d'accuracy obtenez-vous ?

Intéressons nous maintenant à la normalisation des données. Une étape importante du pré processing. Nous allons donc modifier les données de sorte à ce que l'échelle des données entre elles soit comparable. Pour cela nous allons utiliser le module MinMaxScaler disponible avec `from sklearn.preprocessing import MinMaxScaler` et instancions le objet scaler `scaler = MinMaxScaler()`

Une première solution est donnée dans le code suivant

```
X_scaled = scaler.fit_transform(X)

X_train_scaled, X_test_scaled, y_train, y_test = train_test_
split(X_scaled, y, test_size=0.33, random_state=s)

svm.fit(X_train_scaled, y_train)

y_pred = svm.predict(X_test_scaled)
print(accuracy_score(y_test,y_pred) )
```

Une deuxième solution est donnée dans le code suivant

```
X_train_scaled = scaler.fit_transform(X_train)

svm.fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(X_test)

y_pred = svm.predict(X_test_scaled)
print(accuracy_score(y_test,y_pred) )
```

§ Analyser et expliquer en quoi l'approche de normalisation est différente entre ces deux solutions

§ Comparer l'accuracy de ces 3 modèles pour des valeurs de s allant de 1 à 30

Importation des packages nécessaires

In [196]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
X = mybreast_cancer.frame.iloc[:,1:30]
y = mybreast_cancer.frame.iloc[:,30]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

In [197]:
```
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print("Accuracy: %.2f%%" % (accuracy_score(y_test,y_pred) * 100.0))
```

```
Accuracy: 93.09%
```

La valeur d'Accuracy avec ces données est de 93.09%

In [198]:
```
scaler = MinMaxScaler()
```

Première solution

In [199]:
```
X_scaled = scaler.fit_transform(X)

X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_s

svm.fit(X_train_scaled, y_train)

y_pred = svm.predict(X_test_scaled)
print(accuracy_score(y_test,y_pred)*100 )
```

```
98.93617021276596
```

La normalisation est effectuée sur tout le jeu de données avant séparation en train et test et non

Deuxième solution

In [200]:
```
X_train_scaled = scaler.fit_transform(X_train)

svm.fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(X_test)

y_pred = svm.predict(X_test_scaled)
print(accuracy_score(y_test,y_pred)*100)
```

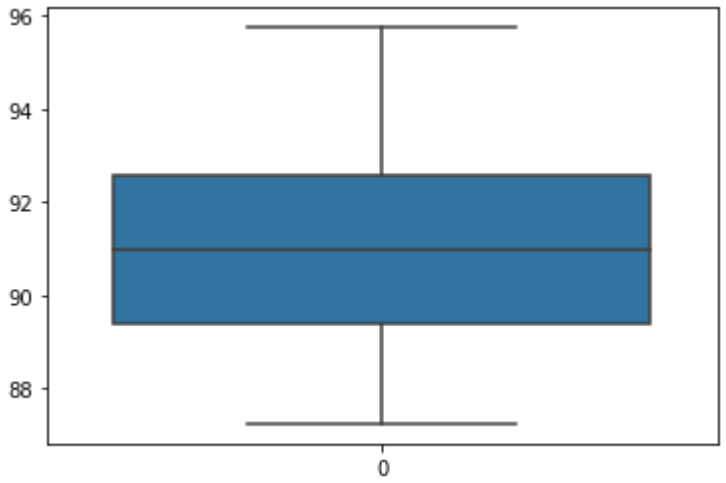
```
98.40425531914893
```

La norme est effectuée sur [illegible] de données de test uniquement

Comparaison des 3 accuracy avec s variant de 1 à 30

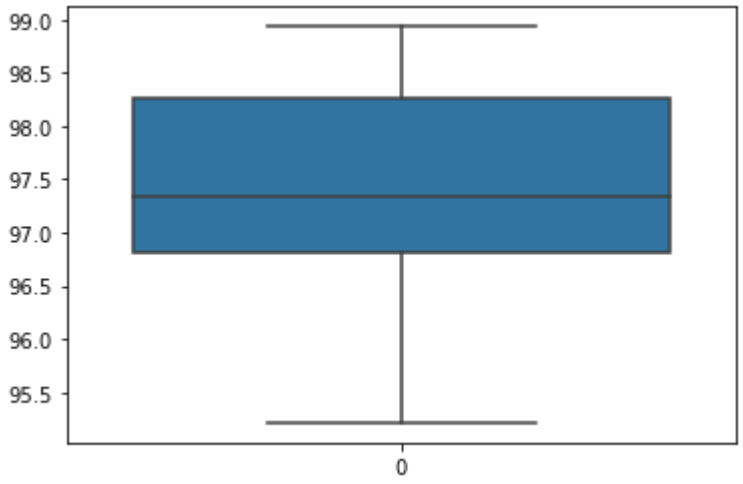
In [201]:
```
L1=[]
for s in range(30):
    X = mybreast_cancer.frame.iloc[:,1:30]
    y = mybreast_cancer.frame.iloc[:,30]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
    svm = SVC()
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    L1.append(accuracy_score(y_test,y_pred) * 100.0)
sns.boxplot(L1)
```

Out[201]:
```
<AxesSubplot:>
```



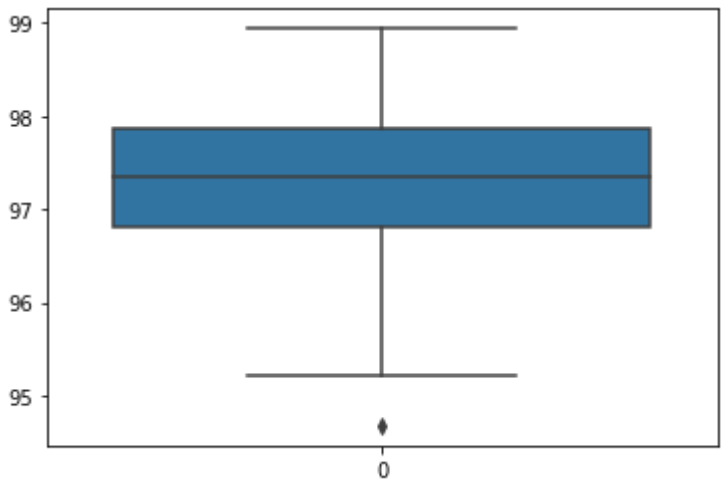
In [202]:
```
L2=[]
for s in range(30):
    X_scaled = scaler.fit_transform(X)
    X_train_scaled, X_test_scaled, y_train, y_test = train_test_split
    svm.fit(X_train_scaled, y_train)
    y_pred = svm.predict(X_test_scaled)
    L2.append(accuracy_score(y_test,y_pred) * 100.0)
sns.boxplot(L2)
```

Out[202]: <AxesSubplot:>



In [203]:
```
L3=[]
for s in range(30):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
    X_train_scaled = scaler.fit_transform(X_train)
    svm.fit(X_train_scaled, y_train)
    X_test_scaled = scaler.transform(X_test)
    y_pred = svm.predict(X_test_scaled)
    L3.append(accuracy_score(y_test,y_pred) * 100.0)
sns.boxplot(L3)
```

Out[203]: <AxesSubplot:>



Conclusion

In [204]:
```
L = [L1,L2,L3]
```

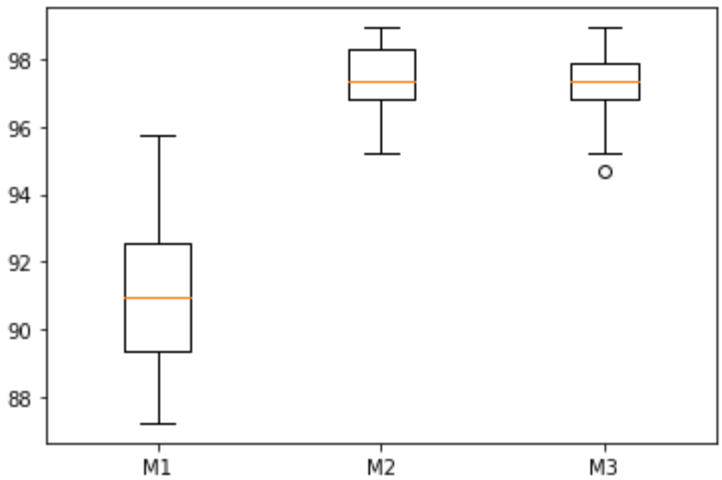
In [205]:

```python
from matplotlib.pyplot import boxplot
data = pd.DataFrame({
    'M1' : L1,
    'M2' : L2,
    'M3' : L3,
})

boxplot(np.array(data), labels = list(data))
```

Out[205]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f6f25cded90>,
  <matplotlib.lines.Line2D at 0x7f6f25c6d130>,
  <matplotlib.lines.Line2D at 0x7f6f25c7a580>,
  <matplotlib.lines.Line2D at 0x7f6f25c7a8b0>,
  <matplotlib.lines.Line2D at 0x7f6f25c84c10>,
  <matplotlib.lines.Line2D at 0x7f6f25c84f40>],
 'caps': [<matplotlib.lines.Line2D at 0x7f6f25c6d490>,
  <matplotlib.lines.Line2D at 0x7f6f25c6d7f0>,
  <matplotlib.lines.Line2D at 0x7f6f25c7abe0>,
  <matplotlib.lines.Line2D at 0x7f6f25c7af10>,
  <matplotlib.lines.Line2D at 0x7f6f25c912b0>,
  <matplotlib.lines.Line2D at 0x7f6f25c915e0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f6f25cdea90>,
  <matplotlib.lines.Line2D at 0x7f6f25c7a250>,
  <matplotlib.lines.Line2D at 0x7f6f25c848e0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f6f25c6db50>,
  <matplotlib.lines.Line2D at 0x7f6f25c84280>,
  <matplotlib.lines.Line2D at 0x7f6f25c91910>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f6f25c6deb0>,
  <matplotlib.lines.Line2D at 0x7f6f25c845b0>,
  <matplotlib.lines.Line2D at 0x7f6f25c91c40>],
 'means': []}
```



Les boxplots montrent une performance plus importante avec normalisation des données (Méthode 1 et 2 ont des médianes d'accuracy et des min et max plus élevés). En ce qui concerne les 2 méthodes de normalisation, la méthode 1 peut sembler plus performante avec une médiane plus élevée. Cependant, en normalisant avant la séparation des données on risque d'effectuer un sur apprentissage des données. Ainsi, d'après cet exercice, il est donc conseillé de normaliser le jeu de données après séparation des données.