Supervised Learning: Support Vector Machines & Naive Bayes Classifier

AAA-Python Edition

Plan

- 1- Support Vector Machines
- 2- Kernel SVM Regressor
- 3- Face recognition with SVM
- 4- Naive Bayes Classification
- 5- Gaussian Naive Bayes
- 6- Multinomal Naive Bayes

1- Support Vector Machines

Principles

- The **support Vector Machine classification** (**SVM**) is a **classification** where the mathematical model is the **optimal hyperplane** that **delimits** the **classes** of the data.

- The **optimal hyperplane**, is the one that **maximizes** the distance from each class.

- In a binary classification, with 2 features, the hyperplane is a **line** that maximizes the distance from the two classes.

- To identify the **optimal** line in this binary classification, a **margin** is drawn **around each** separating line up to the **nearest point** of **each class**.

- The **optimal line**, is the line that **maximizes** this **margin**.

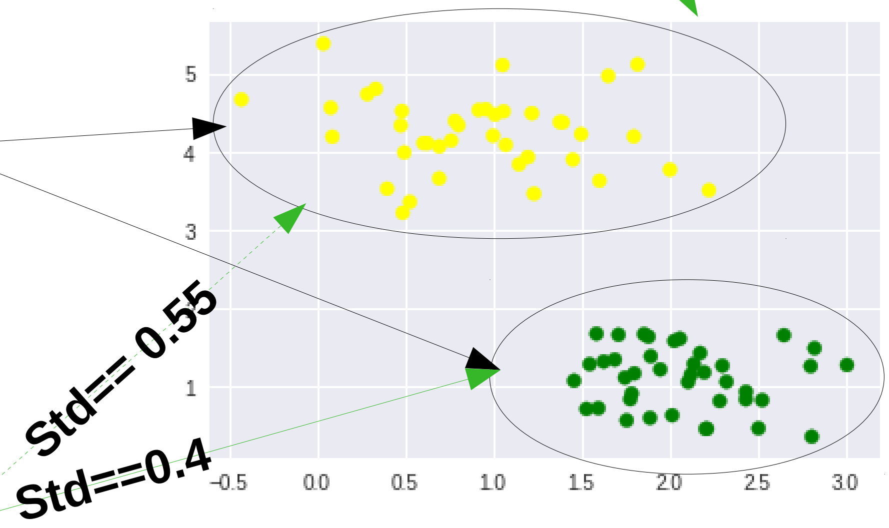
1- Support Vector Machines

Example

```
from sklearn.datasets.samples_generator import make_blobs
# generating random data with 2 classes, standard deviation = 0.55, and random seed ==0
x, y = make_blobs(n_samples=80, centers=2, random_state=0,cluster_std=[0.55,0.4])
# class 0 points are yellow, class 1 points are green
yc= ["green" if i else "yellow" for i in y]
plt.scatter(x[:, 0], x[:, 1],c=yc);
```

This function will generate:

- **80** points
- The points are centered around **2** clusters.
- the seed (a number used by the pseudo random generator. To have different data each time, you have to change the seed at each execution == a random one)== **0**
- The standard deviation (indicates how spares is the data for each cluster)
- The coordinates of these points are in **x :** they represent the **features** values. The labels are in **y**



[By]

1- Support Vector Machines

Example (suite)

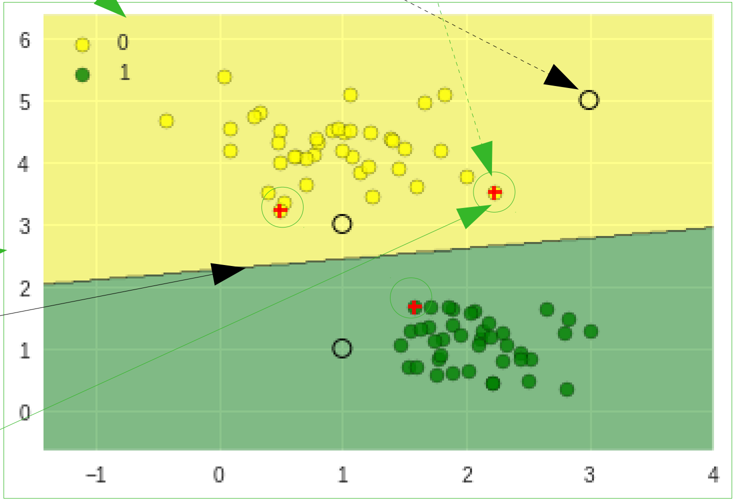
```python
# importing support vector
from sklearn.svm import SVC
# creating an instance of an SVC classifier
myModel =SVC(kernel='linear')
# fiting the model to the data
myModel.fit(x,y)
# the optimal calculated support vectors
vectXY=myModel.support_vectors_
```

Support vector classifier with a linear regressor

```python
# make prediction about some values
toPred=np.array([[1,1],[1,3],[3,5]])
yPred= myModel.predict(toPred)
print("Predicted classes are:\n",yPred)
```

```python
# plotting the results
## plotting the decisions (classes ) regions with the boundary line
## also hilighting the points with predicted classes
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X=x,y=y,clf=myModel, legend=2,colors="yellow,green",
                      markers="oo",X_highlight=toPred)
## plot the suport vector  values
plt.scatter(vectXY[:,0],vectXY[:,1],marker="+",c="red")
```

```
Predicted classes are:
 [1 0 0]
```



The separating line

Support vectors: samples that touch the margins of the classifier

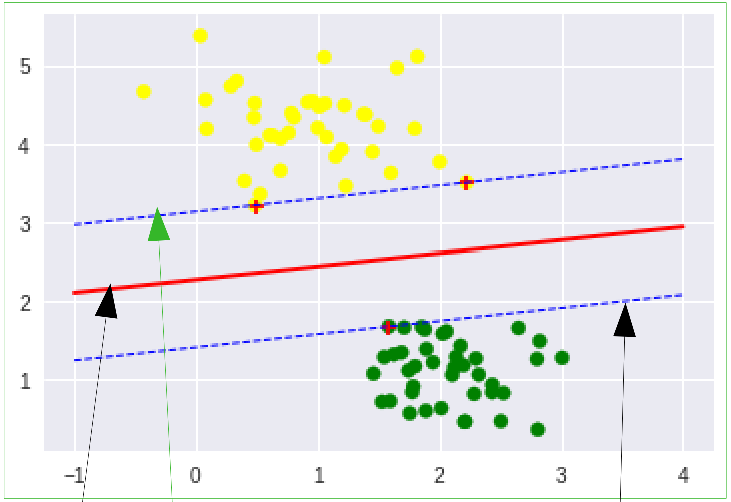
[By]

Deep into SVM

2- Kernel SVM Regressor

```
# the parameters of the regressor
a = myModel.coef_[0]
b = myModel.intercept_[0]
# In general, the linear regressor model is as follow
# y = a1*x1 + a2 *x2 + b
# y represents the distances from the boundary line
# at the boudary line: y == 0
# => a1*x1+a2*x2+b=0 ==> a line's equation
# => x2 = (-a1*x1 - b)/ a2 = -a1/a2 * x1 - b/a2
# we will draw the boundary by initialing x1 values,
# and applying the previous formula, to calculate the
# x2 corresponding values defining the line
x1 = np.linspace(-1,4,1000)
x2=  (- a[0]* x1-b)/a[1]
plt.scatter(x[:, 0], x[:, 1],c=yc);
plt.plot(x1,x2, "r-", linewidth=2)
# plot the suport vector (the red "+" markers)
plt.scatter(vectXY[:,0],vectXY[:,1],marker="+",c="red")
```

```
# plot the margin lines
# the margin lines are paralles to the boundary line
# and their distance from the boundary line |y|== 1
# for the upper line: y==1, for the lower one: y == -1
#upper line equation: ==>a1*x1+a2*x2+b=1 ==> x2==( 1-b-a1*x1)/a2
# lower line equation==>  a1*x1+a2*x2+b=-1 ==>x2==( -1-b-a1*x1)/a2
# we will apply the previous formulas, to calculate the
# xmar1 and xmar2 corresponding values defining the margin lines
x2mar1 =  (1- a[0]* x1-b)/a[1]
x2mar2 =  (-1- a[0]* x1-b)/a[1]
plt.plot(x1,x2mar1, "b--", linewidth=1)
plt.plot(x1,x2mar2, "b--", linewidth=1)
```



[By]

2- Kernel SVM Regressor

Regression Example

```python
# sklearn.datasets utilities doload data samples
# and to use data generators
from sklearn import datasets
# import a support vector machine regressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# shuffle wil
from sklearn.utils import shuffle

#load the data
myData = datasets.load_boston()
# the data is a dictionnary where : the x eatures values are designated by "data" key
# the y labels values are designated by "target" key
myData
```

```
{'DESCR': "Boston House Prices dataset\n===========================\n\nNotes\n------\nData Set
 'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00]
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.09	1.0	296.0	15.3	396.9	4.98

The features

[By]

2- Kernel SVM Regressor

Regression Example (suite)

```
x = myData.data
y = myData.target
# spliting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2
# instantiate a svr with a linear regressor
myModel2 = SVR(kernel='linear')
# training phase
myModel2.fit(x_train, y_train)
# testting the model
yPred = myModel2.predict(x_test)
# Evaluating the model

print( "mean squared error :",mean_squared_error(y_test,yPred))
```

We can use different kernels for regression ==> Kernel SVM

```
mean squared error : 22.74219384815589
```

- In SVM, the model tries to maximizes the margin between the classe
- In an SVR, the model tries to fit as many as possible of samples (points) into that margin.

3- Face recognition with SVM

The data

- The statement **“fetch_lfw_people(min_faces_per_person=10)”** will import a dictionary with the following keys:
  - images: 3-D array with **4324** images. Each image, is described by a 2-D array of **62 rows, 47 columns of pixel** values.
  - data: 2-D array with **4324** samples. Each sample, is described by a 1-D array of **62*47==2914 pixel** values (reshape of “images”).
  - target: labels of the images. Integer **code** indicating the name of the person represented by a data row.
  - target_names: **names** corresponding to codes.

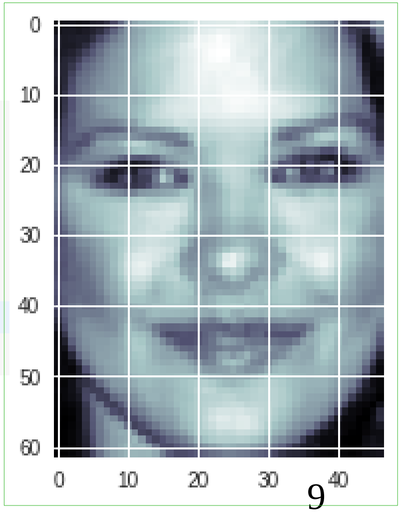
```
# look closer to the images
plt.imshow(faces.images[16],cmap='bone')
print(faces.data[16].shape)
print(faces.images[16].shape)
print(faces.target[16])
print(faces.target_names.shape)
print(faces.target_names[faces.target[16]])
print("images[0,0,0]==",faces.images[0,0,0]," data[0,0]==", faces.data[0,0])
```

```
(2914,)
(62, 47)
21
(158,)
Catherine Zeta-Jones
```

158 people

```
images[0,0,0]== 95.333336  data[0,0]== 95.333336
```

Same values



[By]

3- Face recognition with SVM

Visualizing the data

```
# ploting some images
import matplotlib.image as mpimg
fig, ax = plt.subplots(3, 5)
for i, axi in zip(range(10,25),ax.flat):
    axi.imshow(faces.images[i],cmap="bone")
    axi.set(xticks=[], yticks=[],xlabel=faces.target_names[faces.target[i]][:11])
```

Creating 3*5 == 15 subplots

Returns an iterator over ax: in this case, it's like ax[k,l]

Displaying the images in each subplot

- Disabling the x and y ticks
- xlabel for each subplot corresponds to the person's name whose the face is displayed in that subplot
- we display only 11 characters for each name

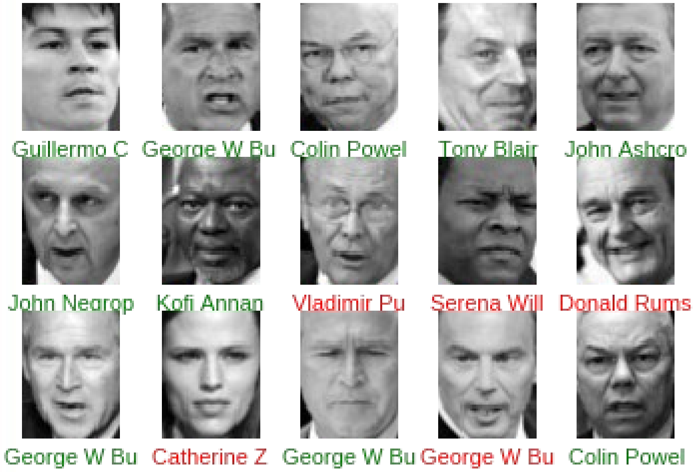
[By]

3- Face recognition with SVM

Training and Testing

```python
# predicting the values
yPred = myModel3.predict(x_test)
# defining the colors for correct and incorrect labels
cY = ["green" if j == i else "red" for i,j in zip(y_test,yPred)]
# ploting some test samples
fig, ax = plt.subplots(3, 5)
for i, axi in zip(range(88,103), ax.flat):
  axi.imshow(x_test[i].reshape(62, 47), cmap='gist_gray')
  axi.set(xticks=[], yticks=[])
  axi.set_xlabel(faces.target_names[yPred[i]][:11],color=cY[i])
```

Displaying some test prediction: the **red** labels are **wrong** labels, the **green** ones are **good**.



	precision	recall	f1-score	support
Abdullah Gul	1.00	0.25	0.40	4
Adrien Brody	0.67	0.50	0.57	4
avg / total	0.50	0.49	0.46	865

The worst results were related to “rbf” kernl. The “poly” kernel (default degree 3) scored piratically same as the “linear” one.

```python
from sklearn.metrics import classification_report
# the classification report
print(classification_report(y_test, yPred,target_names=faces.target_names))
```

[By]

Concept

- **Naive Bayes** models, are a group a of classification algorithms suitable for **high dimensional** datasets.

- They can handle **multiple classes** directly.

- The Naive Bayes classification methods rely on **Bayes's theorem** **(Bayesian classification).**

- In **Bayesian classification**, we try to determine the **probability** of a **label** knowing the **features values.** We note: **P(L|Features)**

- **P(L|features)** is defined by as follow: $$\frac{P(Features|L)\times P(L)}{P(Features)}$$

- To decide between **2** labels, we calculate the ratio:

$$\frac{P(Features|L_1)\times P(L_1)}{P(Features|L_2)\times P(L_2)}$$

[By]

Naive Bayes Classifiers

4- Naive Bayes Classification

- So, we have to use a **model** capable of computing the: **(features|L)**
- This kind of models is called: **generative model:** it is able to generate data for each label.
- We have to define a **generative model** for each label.
- It is difficult to define a "**general"** model ==> so we make **assumptions** about the model ==> we define a **"rough approximation"** of the general model.
- Because of this simplification, we call such classifier: "**Naive Bayes Classifier"**
- The assumption in "**Naive Bayes classifiers** is that **all features** are **independent** given the value of the **class** variable.

[By]

Different Naive Bayes Classifiers

4- Naive Bayes Classification

Gaussian Naive Bayes

- Other assumption == the **data** for each **label** follows a "**simple Guassian Distribution"**
- **==>** the **model** is simply defined by the **mean** and the **standard deviation** of the samples of **each label**

Multinomial Naive Bayes

- Other assumption == the **data** for each **label** follows a "**simple Multinomial Distribution"**
- **==>** the **model** is simply defined by the **probability** of observing **counts**, among a number of **categories**

Gaussian Distribution in scikitlearn

- $$P(x_i \mid y) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma_y^2}} \cdot \exp\left(-\frac{(x_i - \mu_y)^2}{2 \cdot \sigma_y^2}\right)$$

μ_y → Is the features mean value, according to label y (labled y)

σ_y → Is the standard deviation (its square is the variance) mean value, according to label y (labled y)

The two parameters are estimated using **maximum likelihood estimation**

5- Gaussian Naive Bayes

5- Gaussian Naive Bayes

Data

- ```
from sklearn import datasets
iris = datasets.load_iris()
print(iris.keys())
print(iris.DESCR)
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

3 classes and 4 features
```

5- Gaussian Naive Bayes

Training and Testing

```python
from sklearn.model_selection import train_test_split

#import the gaussian naive bayes model
from sklearn.naive_bayes import GaussianNB
myModel4 = GaussianNB()
# spliting the data
x_train, x_test, y_train, y_test =train_test_split(iris.data, iris.target,test_size=0.2)
# training the model
myModel4.fit(x_train, y_train)

# testing the model
yPred= myModel4.predict(x_test)

from sklearn.metrics import classification_report

# print the classification report
print (classification_report(y_test,yPred,target_names=iris.target_names))
```

```
             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        11
 versicolor       0.91      1.00      0.95        10
  virginica       1.00      0.89      0.94         9

avg / total       0.97      0.97      0.97        30
```

[By]

Deep into Multinomial Distribution

6- Multinomial Naive Bayes

- Used in text classification, where the data is represented as vectors of words count.
- It is parametrized by vectors: $\theta_y=(\theta_{y1},\ldots,\theta_{yn})$ for each class y. n is the number of features (size of the vocabulary in a text classification).
- θ_i is P(xi|y) == the probability that the feature i appears in a sample belonging to class y.
- θ_{yi} is computed as follow: $$\hat{\theta}_{yi} = \frac{N_{yi}+\alpha}{N_y+\alpha n}$$

$N_{yi} = \sum_{x \in T} x_i$: : number times that the feature i appears in a sample belonging to class y in the training set T.

- $N_y = \sum_{i=1}^{n} N_{yi}$: : is the total count for all features appearing in class y
- α ≥0 :a smoothing parameter, allows to avoid 0 probability.

[By]

6- Multinomial
Naive Bayes

Data

```
{'data': <11314x130107 sparse matrix of type '<class 'numpy.float64'>'
	with 1787565 stored elements in Compressed Sparse Row format>,
 'target': array([17,  7, 10, ..., 14, 12, 11]),
 'target_names': ['alt.atheism',
  'comp.graphics',
  'comp.os.ms-windows.misc',
  'comp.sys.ibm.pc.hardware',
  'comp.sys.mac.hardware',
  'comp.windows.x',
  'misc.forsale',
  'rec.autos',
  'rec.motorcycles',
  'rec.sport.baseball',
  'rec.sport.hockey',
  'sci.crypt',
  'sci.electronics',
  'sci.med',
  'sci.space',
  'soc.religion.christian',
  'talk.politics.guns',
  'talk.politics.mideast',
  'talk.politics.misc',
  'talk.religion.misc']}
```

```
print(myTextD.data.shape)
```

```
(11314, 130107)
```

If we used: **fetch_20newsgroups**, we will have to vectorize the data, using for example:
from sklearn.feature_extraction.text import TfidfVectorizer

```
from sklearn.datasets import fetch_20newsgroups_vectorized

myTextD = fetch_20newsgroups_vectorized()
myTextD.target_names

myTextD
```


[By]

6- Multinomial Naive Bayes

Training and testing

```python
#import the multinomial naive bayes model
from sklearn.naive_bayes import MultinomialNB
myModel5 = MultinomialNB()
# spliting the data
# spliting the data
x_train, x_test, y_train, y_test =train_test_split(myTextD.data, myTextD.target,test_size=0.2)
# training the model
myModel5.fit(x_train, y_train)

# testing the model
yPred= myModel5.predict(x_test)

from sklearn.metrics import classification_report

# print the classification report
print (classification_report(y_test,yPred,target_names=myTextD.target_names))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.89      0.40      0.55       101
           comp.graphics       0.85      0.54      0.66       127
 comp.os.ms-windows.misc       0.80      0.74      0.77       117
comp.sys.ibm.pc.hardware       0.58      0.81      0.68       113
   comp.sys.mac.hardware       0.95      0.55      0.70       137
          comp.windows.x       0.87      0.81      0.84       113

             avg / total       0.80      0.73      0.71      2263
```

[By]

References

- Aurélien Géron. Hands-on machine learning with Scikit-Learn and Tensor-Flow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc, 2017.

- Joshi Prateek. Artificial intelligence with Python. Packt Publishing, 2017.

- Scikit-learn.org. scikit-learn, machine learning in python. scikit-learn.org/stable/. Accessed on 03-11-2018. On-line at https://scikit-learn.org/stable/.Accessed on 03-11-2018.

- Jake VanderPlas. Python data science handbook: essential tools for working with data. O'Reilly Media, Inc, 2017.

- Harry Zhang. The optimality of naive bayes. American Association for Artificial Intelligence, 1(2):3, 2004.

Thank you!

FOR ALL YOUR TIME