



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- Multinomial 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**.



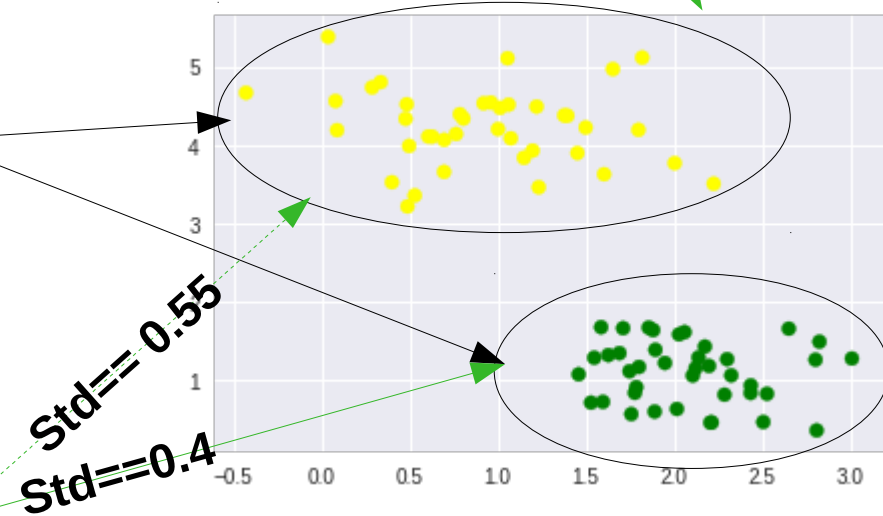
1- Support Vector Machines

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**





1- Support Vector Machines

Example (suite)

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

```
# plotting the results
## plotting the decisions (classes ) regions with the boundary line
## also highlighting 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 support vector values
plt.scatter(vectXY[:,0],vectXY[:,1],marker="+",c="red")
```

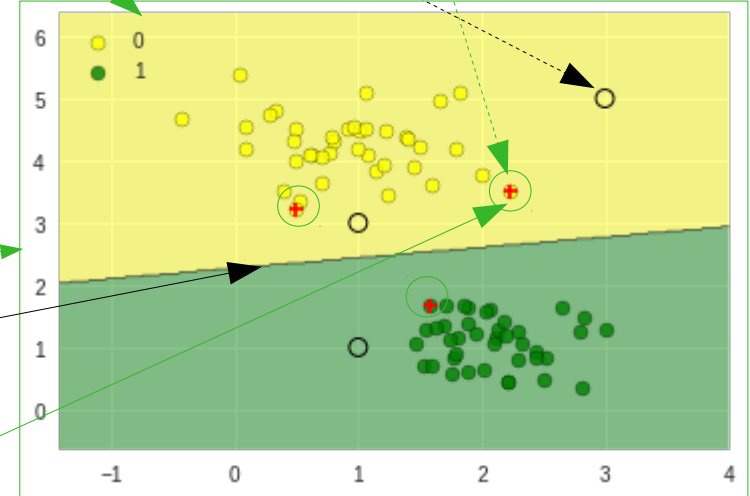
```
1 # importing support vector
2 from sklearn.svm import SVC
3 # creating an instance of an SVC classifier
4 myModel =SVC(kernel='linear')
5 # fitting the model to the data
6 myModel.fit(x,y)
7 # the optimal calculated support vectors
8 vectXY=myModel.support_vectors_
9
```

Support vector classifier with a linear regressor

Predicted classes are:
[1 0 0]

The separating line

Support vectors: samples that touch the margins of the classifier



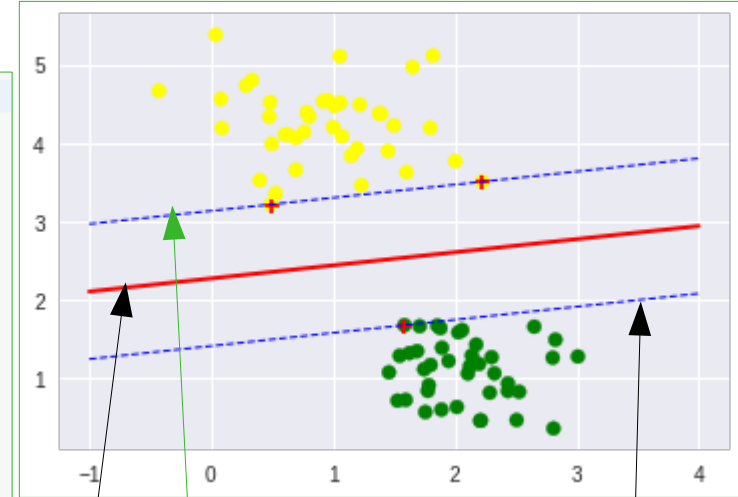


2- Kernel SVM Regressor

Deep into SVM

```
# the parameters of the regressor
a = myModel.coef_[0]
b = myModel.intercept_[0]
# In general, the linear regressor model is as follow
#  $y = a_1x_1 + a_2x_2 + b$ 
#  $y$  represents the distances from the boundary line
# at the boundary line:  $y = 0$ 
#  $\Rightarrow a_1x_1 + a_2x_2 + b = 0 \Rightarrow$  a line's equation
#  $\Rightarrow x_2 = (-a_1x_1 - b) / a_2 = -a_1/a_2 * x_1 - b/a_2$ 
# we will draw the boundary by initialing  $x_1$  values,
# and applying the previous formula, to calculate the
#  $x_2$  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 support vector (the red "+" markers)
plt.scatter(vectXY[:,0],vectXY[:,1],marker="+",c="red")

# plot the margin lines
# the margin lines are parallel 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:  $\Rightarrow a_1x_1 + a_2x_2 + b = 1 \Rightarrow x_2 = (1-b-a_1x_1)/a_2$ 
# lower line equation  $\Rightarrow a_1x_1 + a_2x_2 + b = -1 \Rightarrow x_2 = (-1-b-a_1x_1)/a_2$ 
# we will apply the previous formulas, to calculate the
#  $x_{mar1}$  and  $x_{mar2}$  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)
```





2- Kernel SVM Regressor

Regression Example

```
1 # sklearn.datasets utilities to load data samples
2 # and to use data generators
3 from sklearn import datasets
4 # import a support vector machine regressor
5 from sklearn.svm import SVR
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import mean_squared_error
8 # shuffle will
9 from sklearn.utils import shuffle
10
11 #load the data
12 myData = datasets.load_boston()
13 # the data is a dictionary where : the x features values are designated by "data" key
14 # the y labels values are designated by "target" key
15 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,
               4.0200e+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

[By Amina Delali]

The features



2- Kernel SVM Regressor

Regression Example (suite)

```
1 x = myData.data
2 y = myData.target
3 # splitting the data into training and testing sets
4 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
5 # instantiate a svr with a linear regressor
6 myModel2 = SVR(kernel='linear')
7 # training phase
8 myModel2.fit(x_train, y_train)
9 # testing the model
10 yPred = myModel2.predict(x_test)
11 # Evaluating the model
12
13 print( "mean squared error :", mean_squared_error(y_test, yPred))
14
```

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

mean squared error : 22.74219384815589

- In SVM, the model tries to maximize the margin between the classes
- 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.

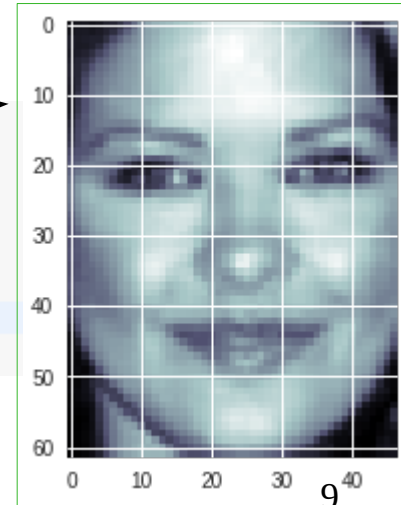
```
1 # look closer to the images
2 plt.imshow(faces.images[16], cmap='bone')
3 print(faces.data[16].shape)
4 print(faces.images[16].shape)
5 print(faces.target[16])
6 print(faces.target_names.shape)
7 print(faces.target_names[faces.target[16]])
8 print("images[0,0,0]==", faces.images[0,0,0], " data[0,0]==", faces.data[0,0])
```

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

Same values

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

158
people





3- Face recognition with SVM

Visualizing the data

Creating $3 \times 5 == 15$
subplots

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

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 Amina Delali]



3- Face recognition with SVM

Training and Testing

```

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

```

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" kernel. The "poly" kernel (default degree 3) scored piratically same as the "linear" one.

```

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

```



4- Naive Bayes Classification

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:

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



4- Naive Bayes Classification

Naive Bayes Classifiers

- 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.



Different Naive Bayes Classifiers

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**



5- Gaussian Naive Bayes

Gaussian Distribution in scikitlearn

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

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

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

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



5- Gaussian Naive Bayes

Data

```
1 from sklearn import datasets
2 iris = datasets.load_iris()
3 print(iris.keys())
4 print(iris.DESCR)
```

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

**3
classes
and 4
features**

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

Training and Testing



5- Gaussian Naive Bayes

```
1 from sklearn.model_selection import train_test_split
2
3 #import the gaussian naive bayes model
4 from sklearn.naive_bayes import GaussianNB
5 myModel4 = GaussianNB()
6 # splitting the data
7 x_train, x_test, y_train, y_test =train_test_split(iris.data, iris.target,test_size=0.2)
8 # training the model
9 myModel4.fit(x_train, y_train)
10
11 # testing the model
12 yPred= myModel4.predict(x_test)
13
14 from sklearn.metrics import classification_report
15
16 # print the classification report
17 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



6- Multinomial Naive Bayes

Deep into Multinomial Distribution

- Used in text classification, where the data is represented as vectors of words count.
- It is parametrized by vectors: $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$ for each class y . n is the number of features (size of the vocabulary in a text classification).
 - θ_i is $P(x_i|y) ==$ 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
 - $\alpha \geq 0$: a smoothing parameter, allows to avoid 0 probability.



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']}]}
```

```
1 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

```
1 from sklearn.datasets import fetch_20newsgroups_vectorized
2
3 myTextD = fetch_20newsgroups_vectorized()
4 myTextD.target_names
5
6 myTextD
```



6- Multinomial Naive Bayes

Training and testing

```
1 #import the multinomial naive bayes model
2 from sklearn.naive_bayes import MultinomialNB
3 myModel5 = MultinomialNB()
4 # splitting the data
5 # splitting the data
6 x_train, x_test, y_train, y_test =train_test_split(myTextD.data, myTextD.target,test_size=0.2)
7 # training the model
8 myModel5.fit(x_train, y_train)
9
10 # testing the model
11 yPred= myModel5.predict(x_test)
12
13 from sklearn.metrics import classification_report
14
15 # print the classification report
16 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
-------------	------	------	------	------



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