IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

UETAF IML

Practical on SVMs

Yannis Haralambous (IMT Atlantique)

October 16, 2019

In this practical we will first scrutinize an elementary example of SVM and then see some SVM applications under Python. The packages used are `scikit-learn` (and its subpackages `svm` et `datasets`), `numpy` and `matplotlib`.

1 An elementary SVM

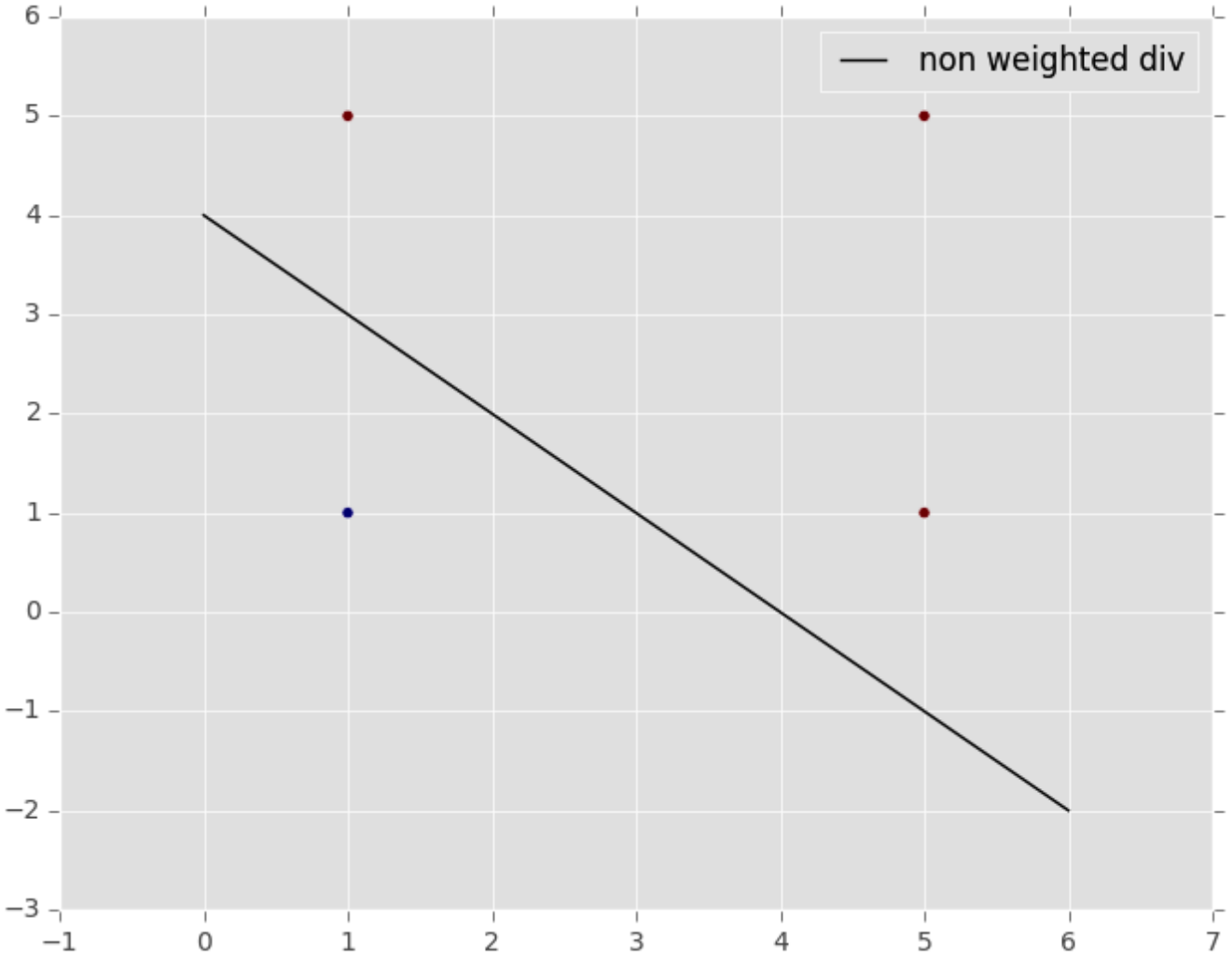
In the Euclidean plane $\mathbb{R}^2$ take individuals $(1,1)$, $(1,5)$, $(5,1)$ and $(5,5)$, assigned to classes $-1,1,1,1$.
1) Manually calculate the vector (a,b) and the value of c of the maximal margin classifier straight line.
2) Implement :

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
X = np.array([[1, 1], [1, 5], [5, 1], [5, 5]])
y = [-1,1,1,1]
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
print clf.support_vectors_
print clf.n_support_

w = clf.coef_[0]
print(w)
a = -w[0] / w[1]
xx = np.linspace(0,6)
yy = a * xx - clf.intercept_[0] / w[1]

plt.plot(xx, yy, 'k-', label="non weighted div")
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.legend()
plt.show()
```

to obtain



What is the meaning of the obtained results :

```
[[ 1.  1.]
 [ 1.  5.]
 [ 5.  1.]]
[1 2]
[ 0.49975586  0.49975586]
```

SOLUTION La droite est antidiagonale donc le vecteur normal est diagonal (a, a). La distance géométrique entre les vecteurs supports et la droite est de $\sqrt{2}$ (Pythagore). On veut que $ax + by + c$ soit 1 aux points $(5, 1)$ et $(1, 5)$. La formule du cours est $d(M, D) = |ax + by + c|/||\alpha||$, donc pour ces deux points, $||\alpha|| = 1/d(M, D) = 1/\sqrt{2}$. Donc le vecteur normal est $(0.5, 0.5)$. Reste à trouver c : $0.5 \cdot 5 + 0.5 \cdot 1 + c = 1 \Rightarrow c = -2$.

2 Iris

We will start with the famous data set "Iris" : the size in centimeters of petals and other parts of some flowers. For every individual we have four number and the class, among*Iris setosa*, *Iris versicolor* and *Iris virginica*. We have 150 individuals and equidistributed classes.

Do

```
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
```

```
iris = datasets.load_iris()
X, y = iris.data, iris.target
```

ut of the iris data create a training set train_X,train_y with 100 random individuals and a test set test_X,test_y with what remains. *Advice : use the shuffle function from package random.*

SOLUTION

```
ind = [i for i in range(150)]
shuffle(ind)
train_X=X[ind[0:100]]
train_y=y[ind[0:100]]
test_X=X[ind[100:]]
test_y=y[ind[100:]]
```

Launch the SVM by writing

```
clf.fit(train_X,train_y)
```

For predictions, use

```
clf.predict(test_X)
```

Calculate, using Python list comprehension the precision and recall of each class.

SOLUTION

```
clf.fit(train_X,train_y)
p=list(clf.predict(test_X))
print (sum([p[i]==test_y[i] for i in range(50)])/50.)*100

for c in range(3):
if sum([p[i]==c for i in range(50)])>0:
print "Precision classe "+str(c)+":"+str(float(sum([p[i]==c for i in range(50) if\
test_y[i]==c]))/float(sum([p[i]==c for i in range(50)])))

for c in range(3):
if sum([p[i]==c for i in range(50)])>0:
print "Rappel classe "+str(c)+":"+str(float(sum([p[i]==c for i in range(50) if\
test_y[i]==c]))/float(sum([test_y[i]==c for i in range(50)])))
```

By using class KFold from package sklearn.cross_validation, write the code a 10-crossed validation and obtain average precision and recall for each class.

Use various kernel types and parameter values to see how the performances vary.

You have the choice between four kernels (option kernel of SVC) :

1. linear linear $k(x, x') = \langle x, x' \rangle$;
2. polynomial pol $k(x, x') = (\gamma \cdot \langle x, x' \rangle + r)^d$;
3. radial rbf $k(x, x') = e^{-\gamma \|x - x'\|}$ (par défaut) ;
4. sigmoid sigmoid $k(x, x') = \text{th}(\gamma \cdot \langle x, x' \rangle)$.

Parameters γ, d and r are written gamma, degree and coef0, resp. The cose parameter C (see in the class) is written cost.

SOLUTION

```
import numpy as np
import matplotlib.pyplot as plt
from random import shuffle
from matplotlib import style
style.use("ggplot")
from sklearn import svm
from sklearn import datasets
from sklearn.cross_validation import KFold
clf = svm.SVC(kernel="poly",degree=2)
iris = datasets.load_iris()
X, y = iris.data, iris.target
ind = [i for i in range(150)]
shuffle(ind)
X=X[ind]
y=y[ind]
kf = KFold(150, n_folds=10)
k=-1
pre=[0.0,0.0,0.0]
rap=[0.0,0.0,0.0]
for train, test in kf:
k+=1
#  print("%s %s" % (train, test))
train_X=X[train]
train_y=y[train]
test_X=X[test]
test_y=y[test]
clf.fit(train_X,train_y)
p=list(clf.predict(test_X))
l=len(test_y)
# print (sum([p[i]==test_y[i] for i in range(l)])/float(l))*100

for c in range(3):
if sum([p[i]==c for i in range(l)])>0:
pre[c] += float(sum([p[i]==c for i in range(l) if test_y[i]==c]))/float(sum([p[i]==c for i in range(l)]))

for c in range(3):
if sum([test_y[i]==c for i in range(l)])>0:
 rap[c] += float(sum([p[i]==c for i in range(l) if test_y[i]==c]))/float(sum([test_y[i]==c for i in range(l)]))

for c in range(3):
print "Classe "+str(c)+" Precision : "+str(pre[c]/10.)+", rappel : "+str(rap[c]/10.)
```

3 Chronic Kidney Disease

3.1 Data preparation

Fetch the file chronic_kidney_disease_full.arff from Moodle. Read the introductory comments and get acquainted with the ARFF data format.

(For more information see http://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease)

Clean up the data. Check that every line has the same number of parameters, if not : correct.

For missing values replace with the average of all existing values in the same column.

Get acquainted with the input format of SVM^{light}, convert the data into that format, save them in a file data.dat.

SOLUTION

```
import os, re
f = open("/Users/yannis/Google_Drive/yannis/texmf/cours/svm/EXAMPLE/Chronic_Kidney_Disease/chronic_kidney_disease
```

```
attribute = dict()
type=dict()
attributes = []
reading_data=0
results=[]
somme = [0] * 25
howmany = [0] * 25

for line in f:
    if (re.match(r'''^@attribute[ ]+'([a-z]+)'[ ]+numeric''',line)):
        m = re.match(r'''^@attribute[ ]+'([a-z]+)'[ ]+numeric''',line)
        name=m.group(1)
        attributes.append(name)
        attribute[name] = dict()
        type[name]='numeric'
    elif (re.match(r'''^@attribute '([a-z]+)' {(.+)}''',line)):
        m=re.match(r'''^@attribute '([a-z]+)' {(.+)}''',line)
        name=m.group(1)
        attributes.append(name)
        values=re.split(',',m.group(2))
        attribute[name] = dict()
        type[name]='modal'
        i=-1
        for x in values:
            i += 1
            attribute[name][x]=i
    elif (reading_data==1):
        vals=re.split(',',line.rstrip())
        if (len(vals)==25):
            results.append(vals)
    elif (re.match(r'''^@data''',line)):
        reading_data=1

for vals in results:
    for i in range(25):
        if (vals[i]!='?'):
            if (type[attributes[i]]=='modal'):
                somme[i] += attribute[attributes[i]][vals[i]]
            else:
                somme[i] += float(vals[i])
            howmany[i] += 1

newresults=[]

for vals in results:
    newvals = [0] * 25
    for i in range(25):
        if (vals[i] == '?'):
            if (howmany[i] > 0):
                newvals[i]=float(somme[i]/howmany[i])
            else:
                newvals[i]=0.0
        elif (type[attributes[i]]=='modal'):
            newvals[i]=float(attribute[attributes[i]][vals[i]])
        else:
            newvals[i]=float(vals[i])
    newresults.append(newvals)
    if (newvals[24]==0):
        result="1"
    else:
```

```
        result="-1"
    for i in range(24):
        result += " "+str(i+1)+":"+str(newvals[i])
    print(result)
```

3.2 Data preparation for cross validation

Read `data.dat` and shuffle lines. Divide in ten parts, put every test corpus into `testi.dat` for $i = 0, \ldots, 9$, and every training corpus into `traini.dat` for $i = 0, \ldots, 9$.

SOLUTION

```
import random, os

f=open("/Users/yannis/Google_Drive/yannis/texmf/cours/svm/data.dat","r")
data=[]

for line in f:
    data.append(line)

random.shuffle(data)

dixieme=int(len(data)/10)+1

for I in range(10):
    train_file = open("train"+str(I)+".dat","w")
    test_file = open("test"+str(I)+".dat","w")
    for J in range(0,I * dixieme):
        if (J < len(data)):
            train_file.write(data[J])
    for J in range(I * dixieme,(I+1) * dixieme):
        if (J < len(data)):
            test_file.write(data[J])
    for J in range((I+1) * dixieme,len(data)):
        if (J < len(data)):
            train_file.write(data[J])
    train_file.close()
    test_file.close()
```

3.3 Cross-validation and results

Download and install SVM$^{\text{light}}$ from `https://www.cs.cornell.edu/people/tj/svm_light/index.html`

Fin out the command line syntax and run SVM$^{\text{light}}$ ten times from within Python on the `testi.dat` and `traini.dat` files. Capture the data returned from SVM$^{\text{light}}$ (hint : use the `subprocess.run` function). Calculate the average runtime, error, precision and recall.

Use various kernels and play with the parameters to see whether the results can be improved, without affecting performance too much.

SOLUTION

```
import subprocess, os, re
runtime=0.0
error=0.0
precision=0.0
recall=0.0
```

```
for I in range(10):
   p = subprocess.run(['svm_learn','-z', 'c', '-c', '1.0', '-t', '1', '-d', '2', 'train'+str(I), 'model'],\
    capture_output=True)
    for line in re.split(r'\n',p.stdout.decode()):
        if (re.match(r'Runtime in cpu-seconds: ([0-9.]+)',line)):
            m=re.match(r'Runtime in cpu-seconds: ([0-9.]+)',line)
            runtime += float(m.group(1))
        if (re.match(r'XiAlpha-estimate of the error: error<=([0-9.]+)',line)):
            m=re.match(r'XiAlpha-estimate of the error: error<=([0-9.]+)',line)
            error += float(m.group(1))
        if (re.match(r'XiAlpha-estimate of the recall: recall=>([0-9.]+)',line)):
            m=re.match(r'XiAlpha-estimate of the recall: recall=>([0-9.]+)',line)
            recall += float(m.group(1))
        if (re.match(r'XiAlpha-estimate of the precision: precision=>([0-9.]+)',line)):
            m=re.match(r'XiAlpha-estimate of the precision: precision=>([0-9.]+)',line)
            precision += float(m.group(1))
        print(str(I)+"...")
print("runtime = "+str(runtime/10)+" cpu seconds")
print("error = "+str(error/10)+"%")
print("precision = "+str(precision/10)+"%")
print("recall = "+str(recall/10)+"%")
```

---

4 SPAM

For those who finished the previous exercise and are still motivated for another example, which they can continue at home, fetch the SPAM dataset from Hewlett-Packard :

`https://archive.ics.uci.edu/ml/datasets/Spambase`

It contains 4 061 individuals with 57 features and the class (SPAM or not SPAM). There are 39,4% SPAM individuals and 60,59% not SPAM individuals. It is preferable to avoid false positives, see if you can get zero false positives and check how many SPAMs pass the filter then.