Support Vector Machines

Saul Garcia

December 10, 2015

Summary

The purpose of this project is to implement and solve a `Support Vector Machine` (svm), with a dataset provided in order to correctly classify our dataset into two different classes. In order to accomplish this, we will consider m points of $\mathbb{R}^n$, represented by the matrix $A \in \mathbb{R}^{mxn}$.

Intro

The support vector machine consists in a binary classification method using hyperplanes. The objective is to separate correctly the two classes in the dataset which in this case are labeled -1 or $+1$, by minimizing the distance within the classes, while maximazing the separation between them.

In order to reach a solution for this problem, a standard SVM with a lineal kernel will be utilized, which is characterized by the following quadratic optimization problem for a parameter $\nu > 0$:

$$\begin{aligned} \underset{(w,\gamma,y)\in\mathbb{R}^{n+1+m}}{min} \quad & \nu e^T y \ + \ \frac{1}{2} \ ||w||_2^2 \\ s.t \quad & D(Aw - e\gamma) + y \geq e \\ & y \geq 0 \end{aligned}$$

Building the model

In order to build the model, the dataset consists in a dataframe of size (1000 x 4), which are divided into training and a testing sets with 80% and 20% respectively. The labels will consist in a vector of (1000 x 1) which is also divided in the same proportion.

The training data is divided into `instances` being our matrix `A`, and our `labels` which are `D`, a diagonal matrix. The value of `e` is to be defined as a m long vector of dimensions (m x 1) and `v` has been defined as 10 in order get a better prediction.

In order to use the `CVXPY Solver`, setting `y`, `gamma` and `w` need to be define as variables with the respective dimensions (m x 1), (1 x 1), (n x 1).

After running the function through the solver, is now possible to define our classification model according to the following:

$$\begin{aligned} x^T \ w - \gamma + y_i \geq +1, \quad for \quad x^T = A_i. \quad and \quad D_{ii} = +1 \\ x^T \ w - \gamma + y_i \leq +1, \quad for \quad x^T = A_i. \quad and \quad D_{ii} = -1 \end{aligned}$$

Prediction

For the purpose of prediction, the data of the testing set will be utilized as an input for the model built. Since the model was built with the 80% percent of the data, this means that our 20% has smaller dimensions. For this specific reason the y_i output value in the formula will be considered as a 0s vector, and the threshhold function will classify the output as:

$$+1 \quad for \quad output \geq 1$$
$$-1 \quad otherwise$$

Results

The model is able to produce the following results represented in a Confusion Matrix.

Confusion Matrix

Experiment 1	Prediction Positive	Prediction Negative
Actual Positive	85	7
Actual Negative	8	100

With this information, is now possible to measure the error for the model:

The `Accuracy` given by the model is of 92.5%, which represents the percentage of correct guesses for the given model. But since it could be misleading depending on the amount of information for each class, the precision, recall and F measure have still to be calculated.

The `Precision` given by the model is of 91.4%, this represents the percentage of positive classified values that were actually relevant to the test.

The Recall, with a value of 92.4%, or also known as *Sensitivity*, express how good the model is at detecting the positive classified values.

Finally, the F-score 91.9%, which conveys the balance between precision and recall.

Conclusion

The Support Vector Machine is a simple and efficient method to implement for supervised machine learning, when classifying the data into two classes. Even by having some missclassified points in the inital data, the alrogithm proved to be good enough to deliver a remarkable performance while assuring with a certainty over 90% that each point is well classified.

Python Code

Importing the packages

```
# Import packages
import pandas
from cvxpy import *
import numpy as np
import math
import matplotlib.pyplot as plt
```

Function

```
def threshold(x):
    if x>=0:
        return 1
    else:
        return -1
thresholdfunc =np.vectorize(threshold)
```

Importing Data

```
#Training Data
trainx = np.genfromtxt('data.txt',delimiter=" ",usecols=[0,1,2,3])[0:800,]
trainy = np.genfromtxt('data.txt',delimiter=" ",usecols=[6])[0:800,]
#Testing Data
testx = np.genfromtxt('data.txt',delimiter=" ",usecols=[0,1,2,3])[800:1000,]
testy = np.genfromtxt('data.txt',delimiter=" ",usecols=[6])[800:1000,]
```

Variables and Constants

```
#Input for the optimization
A=trainx                    #input training data
D=np.diag(trainy)           #labels training data
[m,n] = np.shape(trainx)    #dimensions
e=np.ones(m)
v= 10

#Variables to optimize
y= Variable(m)
gamma=Variable()
w= Variable(n)
```

Solver

```
#Problem
objective = Minimize(v*e.T*y + .5*power(norm(w),2))
constraints = [D*(A*w-e*gamma)+y >= e, y>=0]
prob = Problem(objective, constraints)
prob.solve()
```

Predicting “‘python #Prediction on Testing Dataset A_test=testx

model= A_test*w.value - gamma.value pred= thresholdfunc(model) “ Mesuring Error

```
%matplotlib inline
from sklearn import metrics
from sklearn.metrics import confusion_matrix

print "Accuracy:", metrics.accuracy_score(testy, pred)

cm = confusion_matrix(testy, pred)

plt.matshow(cm)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

Confusion Matrix

```
print(cm)
```

Ratios of Confusion Matrix

```
TP=float(cm[0,0])
FN=float(cm[0,1])
FP=float(cm[1,0])
TN=float(cm[1,1])
Accuracy = (TP + TN)/ (TP+FN+FP+TN)
Precision= TP/(TP+FP)
Recall = TP/(TP+FN)
F1=2*(Precision*Recall)/(Precision+Recall)

Accuracy, Precision, Recall,F1
```