SVM Tutorial - Part I

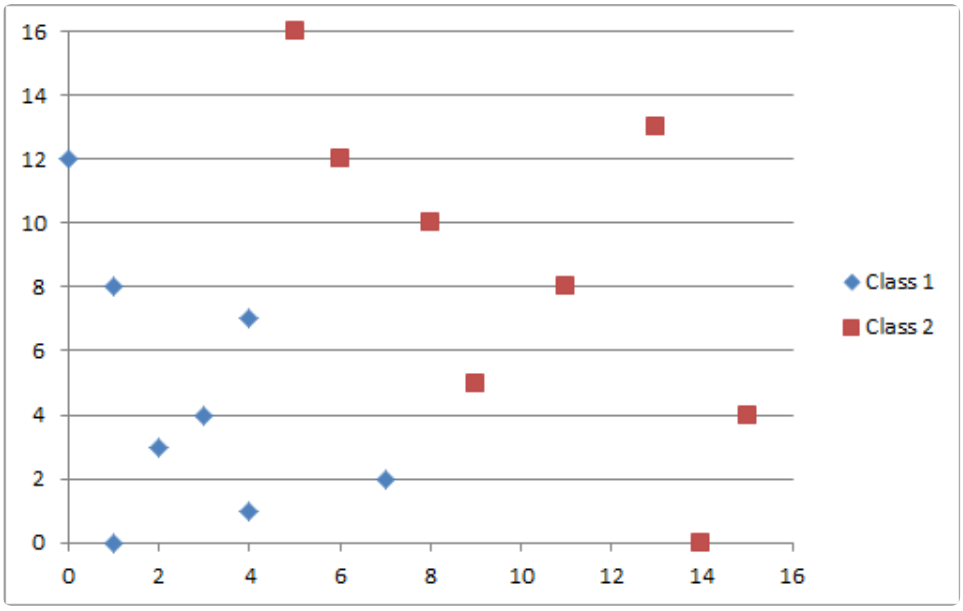
16 Apr 2013

I found it really hard to get a basic understanding of Support Vector Machines. To learn how SVMs work, I ultimately went through Andrew Ng's Machine Learning course (available freely from Stanford). I think the reason SVM tutorials are so challenging is that training an SVM is a complex optimization problem, which requires a lot of math and theory to explain.

However, just by looking at an SVM that's been trained on a simple data set, I think you can gain some of the most important insights into how SVMs work.

I fabricated a very simple SVM example in order to help myself understand some of the concepts. I've included the results and illustrations here, as well as the data files so that you can run the algorithm yourself.

In Excel, I created two sets of points (two "classes") that I placed arbitrarily. I placed the points such that you can easily draw a straight line to separate the two classes (the classes are "linearly separable"). These points are my training set which I used to train the SVM.



Here is the Excel spreadsheet containing the data values and the plots in this post.

SVM Scoring Function

A trained Support Vector Machine has a scoring function which computes a score for a new input. A Support Vector Machine is a binary (two class) classifier; if the output of the scoring function is negative then the input is classified as belonging to class y = -1. If the score is positive, the input is classified as belonging to class y = 1.

Let's look at the equation for the scoring function, used to compute the score for an input vector *x*.

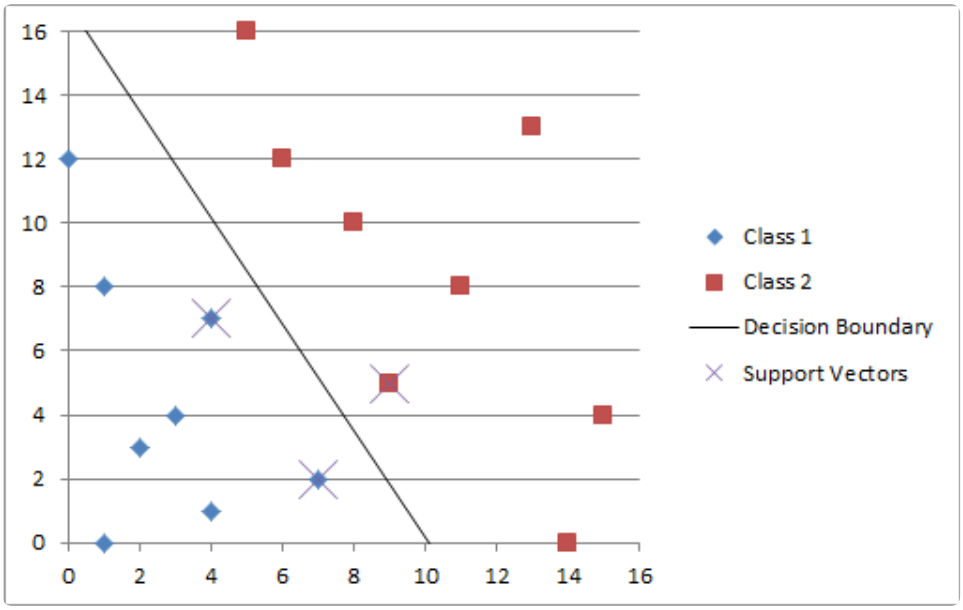
$$\sum_{i=1}^{m} \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + b$$

- This function operates over every data point in a training set (i = 1 through m).
    - Where *x*^(i), *y*^(i) represents the _i_th training example. (Don't confuse this as "x to the ith power")
    - *x*^(i) is an input vector which may be any dimension.
    - *y*^(i) is a class label, which has one of only two values, either -1 or 1.
    - α_i is the coefficient associated with the _i_th training example.
- *x* is the input vector that we are trying to classify
- *K* is what is called a kernel function.
    - It operates on two vectors and the output is a scalar.
    - There are different possible choices of kernel function, we'll look at this more later.
- *b* is just a scalar value.

Support Vectors

This scoring function looks really expensive to compute. You'll have to perform an operation on every single training point just to classify a new input *x*–what if your training set contains millions of data points? As it turns out, the coefficient α_i will be zero for all of the training points except for the "support vectors".

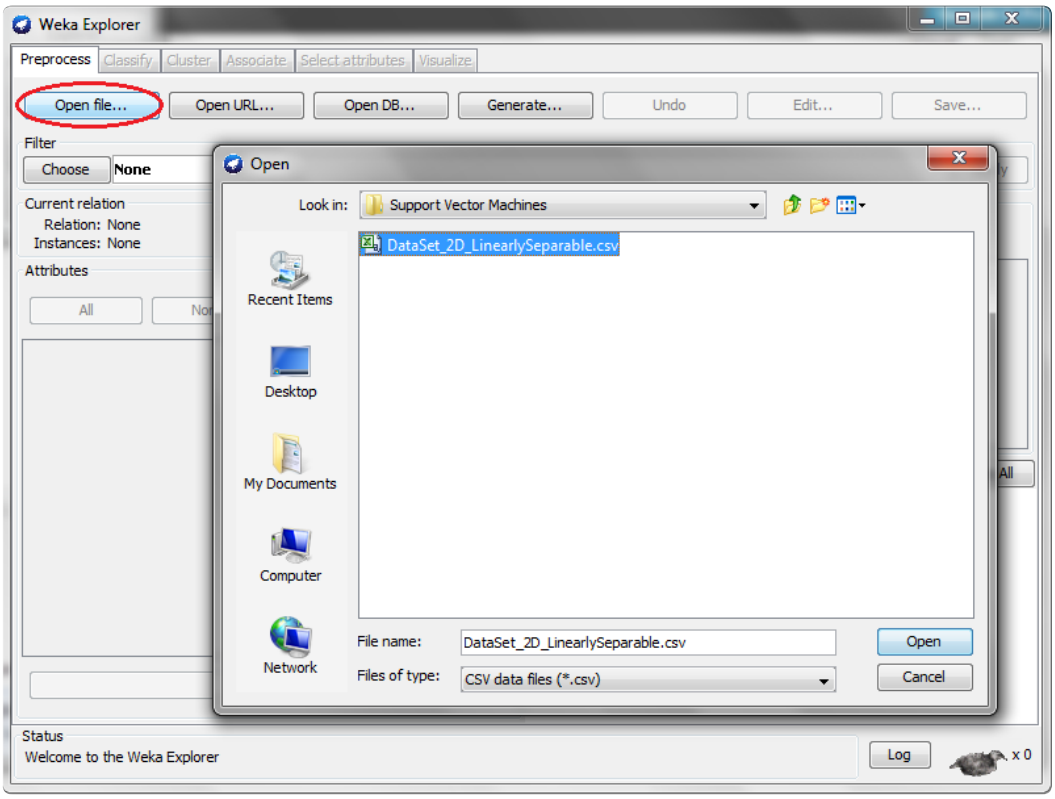
In the below plot, you can see the support vectors chosen by the SVM–the three training points closest to the decision boundary.



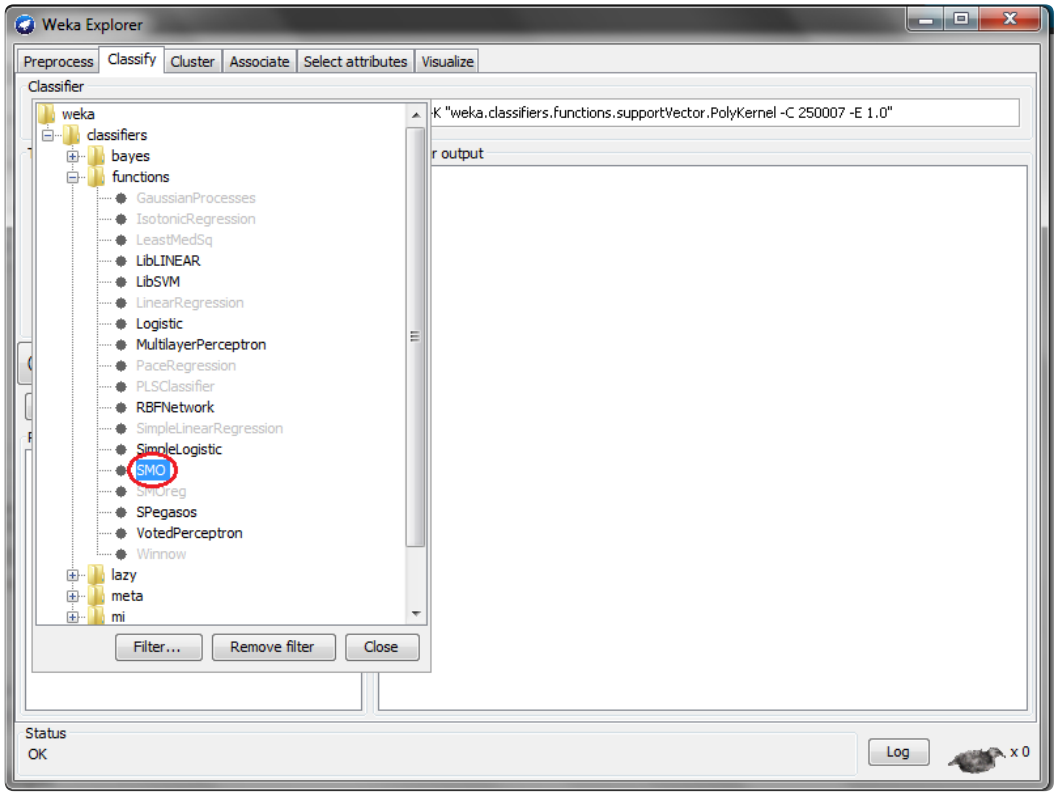
Training The SVM In WEKA

To train an SVM on this data set, I used the freely available WEKA toolset.

1. In the WEKA explorer, on the 'Preprocess' tab, open this .csv file containing the data set.



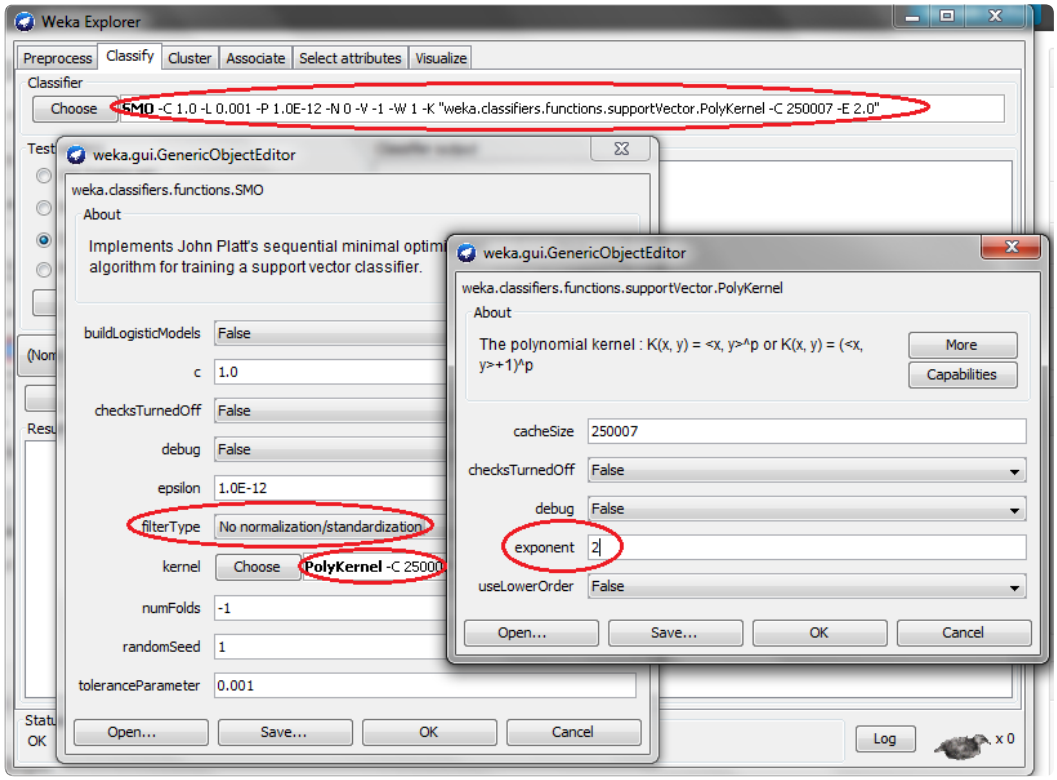
1. On the 'Classify' tab, press the "Choose" button to select classifier weka->classifiers->functions->SMO (SMO is an optimization algorithm used to train an SVM on a data set).



1. Click on the classifier command line to bring up a dialog for editing the command line arguments.

In the dialog, change the 'filterType' property to "No normalization/standardization". This will make the results easier to interpret.

Also, click the command line of the 'kernel' property. This will bring up another dialog to allow you to specify properties of the kernel function. Set the 'exponent' property to 2.



A note for those who are already familiar with kernels: Since our data set is linearly separable, we don't really need an exponent of 2 on the kernel. This is necessary, though, to force WEKA to use support vectors. Otherwise, it will just give you a simple linear equation for the scoring function, which doesn't help us in our understanding of SVMs.

1. Click 'Start' to run the training algorithm.

Towards the middle of the output, you should see something like the following equation:

```
      0.0005 * <7 2 > * X]
-     0.0006 * <9 5 > * X]
+     0.0001 * <4 7 > * X]
+     2.7035
```

This is essentially the scoring function that we saw at the beginning of the post, but now with the values filled in.

The numbers in angle brackets are our three support vectors <7 2>, <9 5>, and <4 7> (these are the points I marked in the scatter plot).

The coefficient beside each support vector is the computed 'alpha' value for that data point.

The sign of the coefficient comes from the class label. For example, <9 5> belongs to class y = -1, and <7 2> belongs to class y = 1. In the original scoring function, there was the term α_i * *y*^(i). The alpha values will always be greater than or equal to 0, so the sign of the coefficient comes from the class label *y*^(i).

The final value in the expression, 2.7035, is *b*.

Visualizing The Scoring Function

Now we can take our scoring equation:

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + b$$

And plug in the values we've found to get:

$$(1)(0.0005)(7x_1 + 2x_2)^2 + (-1)(0.0006)(9x_1 + 5x_2)^2 + (1)(0.0001)(4x_1 + 7x_2)^2 + 2.7035$$

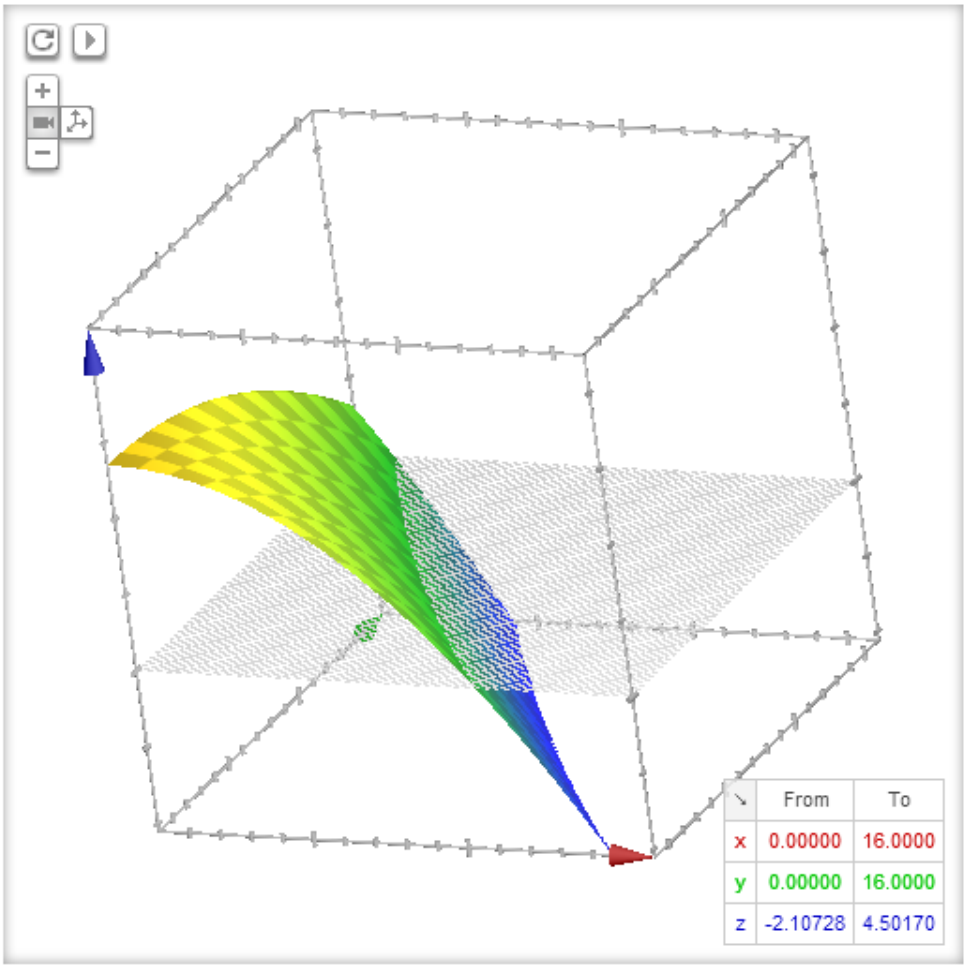
Where x1 and x2 are the components of the input vector _x _that we want to compute a score for.

Note that the original summation was over every point in the training set, but we've only included the terms for the support vectors here. Alpha is zero for all of the other data points, so those terms disappear.

You can plot the above equation using Google; paste the following into a Google search bar:

plot z = 0.0005*(7x + 2y)^2-0.0006*(9x+5y)^2+0.0001(4x+7y)^2+2.7035

Change the ranges to x: 0 - 16 and y: 0 - 16 and you should get something like the following:



The scoring function forms a surface in three dimensions. Where it intersects the z = 0 plane it forms a line; this is our decision boundary. To the left of the decision boundary, inputs receive a score higher than 0 and are assigned to class y = 1. To the right inputs receive a score less than 0 and are assigned to class y = -1.

In the next example, we'll look at a slightly more complex classification problem where the classes are not linearly separable. In that example, we'll go into more detail about the kernel function and how it's used to achieve non-linear classification.

files?

21 • Reply • Share ›

Greg Jonason ➚ SANTOSH KUMAR BANBHRANI • a year ago

Santosh,
You can take a look at any .arff file and conform the data in your .txt files to that format. There are plenty of .arff files online or in the "weka-#-#-#" folder that comes with the Weka installation.

• Reply • Share ›

Chris McCormick Mod ➚ SANTOSH KUMAR BANBHRANI • 2 years ago

I haven't used WEKA in a long time, so I may not be able to help. But a "CSV" file is pretty straightforward, it stands for "comma separated values". It just means each value is separated by a comma, and each datapoint is on its own line.

• Reply • Share ›

Phuc Coi • a minute ago

where is part 2 of SVM?

• Edit • Reply • Share ›

BIRESWAR BANIK • 2 months ago

Thanks a lot..its very easy to understand from this example that how SVM works and how can we implement it. Its truly helpful.Using the scoring mechanism, we can have the graph too.
But I am not getting that how can we obtain the graph in Weka like your 2nd diagram(graph) in this page under the heading Support Vectors.How can we obtain this type of 2D graph from the result that we get after implementing SVM in weka?

• Reply • Share ›

Aditya Gupta • 7 months ago

You just saved my day !! Thanks
But where is part II
One confusion- What are kernel methods, if they are the learning algorithm then what is SVM and in the above example which one actually works?

• Reply • Share ›

Chris McCormick Mod ➚ Aditya Gupta • 7 months ago

The kernels are modifications to the model which make it non-linear. The training method remains the same.

The purpose of kernels, like the Gaussian kernel, is to make the model able to express more complex boundaries between classes. An SVM with a Gaussian kernel is also referred to as an "RBF SVM". It's more expensive to train and evaluate, and it needs more training data to work effectively without overfitting. But if you have enough data it can deliver higher accuracy.

• Reply • Share ›

Haaris Ch • 2 years ago

Hi , Very informative tutorial .. where is the second part with non-linear data ??

please add this too i am currently working on svm and this was helpful

• Reply • Share ›

Chris McCormick Mod ➚ Haaris Ch • 2 years ago

Hi, Haaris - Thanks for the comment, glad it was helpful. Unfortunately, I don't plan to do part II anytime soon... Sorry! :(

One of the most typical non-linear kernels for SVMs is the Gaussian kernel. When you use this kernel, the resulting model is often referred to as an RBF SVM. The resulting model is the same as an RBF Network, it's just that it's been trained using the SVM algorithm. So, you could take a look at my tutorials on RBF networks to learn a little more about what a RBF model looks like in the end (Just know that the training process is different for an RBF SVM).

• Reply • Share ›

Ruben Peralta ➚ Chris McCormick • a year ago

:(I really wanted to see the second part

• Reply • Share ›

Related posts

Product Quantizers for k-NN Tutorial Part 2 22 Oct 2017
Product Quantizers for k-NN Tutorial Part 1 13 Oct 2017
k-NN Benchmarks Part I - Wikipedia 08 Sep 2017