

Jeremy's Blog (<http://mcminis1.github.io/>)

Data Science, Physics, and Randomness

- Atom (</atom.xml>)

- Archives (</archives.html>)
- Home Page (<http://scalingsolutions.net>)

Intuition for Support Vector Regression and the Gaussian Process

May 10, 2014

There is a lack of introductory to intermediate material on using vector machines on the regression problem. In this post, we'll talk about the theory behind the method and try to provide some intuition for it. In a following post, I'll give some code examples and illustrate how the different pieces fit together. I hope that by the end of this series of posts you'll understand how to build, tune, and use a support vector regression (SVR) machine. Because of my personal preference, I'll generally focus on a variant of the SVR known as a Gaussian process (GP), and comment on any generalizations necessary to transfer that knowledge to SVR.

Brief introduction to SVR and GP

Support vector regression (SVR) is a fast and accurate way of interpolating data sets. It is useful when you have an expensive function you want to approximate over a known domain. It learns quickly and is systematically improvable. Variants of SVR are used throughout science including Kriging and Gaussian processes (GP).

Support vector regression is a generalization of the support vector machine to the regression problem. Technically, it can be labelled as a supervised learning algorithm. It requires a training set, $\mathcal{T} = \{ \vec{X}, \vec{Y} \}$ which covers the domain of interest and is accompanied by solutions on that domain. The work of the SVM is to approximate the function we used to generate the training set, $f(\vec{X}) = \vec{Y}$. In my mind, it's just an interpolation scheme.

In a classification problem, the vectors \vec{X} are used to define a hyperplane that separates the two different classes in your solution. In SVR, these vectors are used to perform linear regression. The vectors closest to your test point, or decision boundary are the ones we refer to as support vectors. For regression, it turns out that all of the vectors are support vectors. We can evaluate our function anywhere, so any vectors could be closest to our test evaluation location.

Gaussian processes are a particular form of SVM. The difference between the two lies in choice of kernel and in loss function. The functional form of the kernel determines which vectors in your training set most strongly influence the regression and the form of your estimator. The loss function choice determines the coefficients used in regression. Together these two pieces totally determine the form and accuracy of your estimator. Although this makes it sound like the two are totally different, the spirit of the two are identical. Hopefully, that point will come across as I go over the specifics of them both.

How do you build a SVR?

All right. Let's get right to the point and define how you would go about building a SVR. To build a SVR you must perform 4 tasks,

1. Collect a training set, $\mathcal{T} = \{ \vec{X}, \vec{Y} \}$.

2. Choose a kernel and its parameters as well as any regularization you may need.
3. Form the correlation matrix, \bar{K} .
4. Train your machine, exactly or approximately, to get contraction coefficients, $\vec{\alpha} = \{\alpha_i\}$.
5. Using those coefficients, create your estimator, $f(\vec{X}, \vec{\alpha}, x^*) = y^*$.

Sounds like a lot of work? It's not! Just wait till you see the code in our next installment. For now, let's take each of these steps and talk a little bit about what they are doing.

The training set

Your training set consists of the samples you've collected to train your machine. They should span the domain you are expecting to evaluate later. The SVR is only able to interpolate in between data that exists in your training set. For instance, if you are using your SVR machine to estimate the cosine function, you should train it on the domain $(0 \rightarrow 2\pi)$ (and shift your evaluations into that range). When you evaluate the SVR machine outside the domain you evaluate the result will vary depending on your choice of kernel and optimization routine.

Choosing a kernel

You are free to choose whatever kernel you would like. In this post and the following example I'll focus on using the Gaussian kernel, $K_G(\vec{x}^i, \vec{x}^j, \vec{\theta}) = \exp\left(-\sum_k \theta_k \left|x_k^i - x_k^j\right|^2\right)$ where (N_D) is the number of dimensions in each data point \vec{x} and the vector $\vec{\theta}$ is a set of hyperparameters. Some other common choices of kernel are polynomial, Laplacian, sigmoid, and radial basis functions. All of these functions have a set of hyperparameters that must be trained.

In the gaussian process framework, the statistical interpretation of these parameters are somewhat complicated, but totally worth checking out. I refer you to one of my favorite papers on the subject, "[Bi-fidelity fitting and optimization](http://scitation.aip.org/content/aip/journal/jcp/136/7/10.1063/1.3684884)" (<http://scitation.aip.org/content/aip/journal/jcp/136/7/10.1063/1.3684884>), if you can get behind the paywall. Otherwise a pretty decent source can be found at "[Gaussian Processes for Machine Learning](http://www.gaussianprocess.org/gpml/chapters/)" (<http://www.gaussianprocess.org/gpml/chapters/>). In short, they are chosen such that they maximize the marginal probability of the training set being generated using the kernel as the Bayesian prior. This is because the loss function we use in the hyperparameter optimization step is the log probability of the parameters.

This training is the most expensive part of performing SVR, and much research has gone into developing good ways to do it. We can train it straightforwardly, if expensively, using CG or another gradient based optimization method and minimizing the cost function. The explicit form of the cost function can be found in equation 5.8 in "[Gaussian Processes for Machine Learning](http://www.gaussianprocess.org/gpml/chapters/)" (<http://www.gaussianprocess.org/gpml/chapters/>). Other forms of SVM will use different cost functions and parameter selection routines.

Does the kernel matter?

Yes! If you choose a kernel that goes to zero as the distance between its arguments grows, such as the Gaussian above, $K_G(0, \infty) = 0$, then as you move away from your training data, the machine will return the mean value of your training set. If you choose a function that grows larger, such as linear or polynomial, the value will also grow linearly or polynomially. Your choice of kernel will determine the asymptotic behavior of your estimator. Choose your kernel appropriately for your problem.

Does the regularization matter?

Yes! For training sets with some noise, the regularizer helps prevent wild fluctuations between data points by smoothing out the prior. You can think of this process as adding some slack to the machine. Regularization is achieved by adding a small positive component to the diagonal of the correlation matrix.

Forming the correlation matrix

Forming this matrix is straightforward and really doesn't require any discussion. You evaluate your kernel for all pairs of points in the training set and then add your regularizer to the diagonal. This results in the matrix, $K_{i,j} = \exp\left(-\sum_k \theta_k \left|x_k^i - x_k^j\right|^2\right)$

$\text{right}\} + \epsilon \delta_{i,j}$ where ϵ is your regularizer and $\delta_{i,j}$ is the Kroeniker delta function. This matrix is symmetric positive definite (semi-definite in the numerical sense) and easy to work with. It represents the correlation matrix of the data in a higher dimensional space (<http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf>) than the one your training set is drawn from.

Solve the matrix to form your estimator

This is the meat of the algorithm. Fortunately, it's all just linear algebra. Your machine works like this, $\bar{K} \alpha = y$ where y is the vector of values corresponding to your training set, \bar{K} is still your correlation matrix, and α is a set of unknowns we need to solve for. Doing so is simple, we just invert the matrix \bar{K} and apply it to the vector, y , $\alpha = \bar{K}^{-1} y$. Because our correlation matrix is so well behaved, efficient methods for inversion can be used (QR/Cholesky).

Forming the estimator

Once your α parameters are known, forming the estimator is straightforward. We use the coefficients we found during the optimization step and the kernel we started off with. To estimate the value, y^* , for a test point, x^* , we simply compute the correlation vector, k , and find the inner product with the alpha vector, $y^* = \alpha \cdot k$ where we compute the elements of k as, $k_i = \exp(-\sum_k \theta_k |x_k^i - x_k^*|^2)$.

You can see from the previous equation, if our regularizer is zero and we've inverted our matrix exactly, then the estimator perfectly passes through all of our training points. That is because the vector k is identical to a row in the correlation matrix if the test point is in the training set! Once the regularizer is included or we have gotten our parameters using something quick but approximate, we don't recover our training set solutions, but they should be close.

Dirty tricks

You may notice that I have implicitly removed the mean value from my training set data. You add the mean value from the training set back into the estimator when you're done, $y^* = \alpha \cdot k + \mu$ where $\mu = \bar{y}$ and all y are recentered, $y \rightarrow y - \bar{y}$. This makes it easier for the algorithm to interpolate the differences between the data points in your training set instead of working very hard to also reproduce the mean.

In practice, you also want to scale your inputs so they all have the same range. The reason for this trick is the same as for any other machine that requires some nonlinear optimization. You're trying to make the minimum for your cost function as smooth and harmonic looking as possible.

What is the SVR *really*?

So, now that I've taken all this time to tell you how a SVR machine, and in particular Gaussian process machine, is built, I owe you an intuitive explanation of what it is doing. It's not an easy task, so bear with me for a bit and I hope it will pay off.

The SVR performs linear regression in a higher (infinite) dimensional space. A simple way to think of it is as if each data point in your training set represents its own dimension. When you evaluate your kernel between a test point and a point in your training set, the resulting value gives you the coordinate of your test point in that dimension. The vector we get when we evaluate the test point for all points in the training set, k , is the representation of the test point in the higher dimensional space. The form of the kernel tells you about the geometry of that higher dimensional space.

Once you have that vector, you use it to perform a linear regression. You can tell it is a linear regression because of the form of the estimator, it's an inner product! Thus the intuition for the machine is rather simple, though the procedure and parameters may be difficult to interpret.

So, what's the SVR good for?

Because you are able to generate training points, you know what the "right" answer is. However, it may be very expensive to compute that answer for every new point you need. The SVR, and in particular Gaussian processes, are very good at providing a cheap surrogate to an expensive call. If the function you are trying to compute is smooth, and you expect to have to call it over and over again, you may be able to gain a significant savings by pre-computing a training set and then using a SVR machine to interpolate the results.

Further reading

For a more extensive discussion and tutorial on SVR you should check out Smola and Scholkopf's excellent review from several years back, "[A Tutorial on Support Vector Regression \(http://alex.smola.org/papers/2003/SmoSch03b.pdf\)](http://alex.smola.org/papers/2003/SmoSch03b.pdf)." You can also find more detail in texts such as "[The Elements of Statistical Learning \(http://statweb.stanford.edu/~tibs/ElemStatLearn/\)](http://statweb.stanford.edu/~tibs/ElemStatLearn/)."

Next time

At this point we'll split the blog postings up. I'll talk a little about the parameters and give an example. Ben will put things into context by talking about some of the different SVR machines and related pieces. John will take my learning speed and benchmark it against a much faster version built using continuum's tools.

Drop Jeremy a line at [Quora \(http://www.quora.com/Jeremy-McMinis\)](http://www.quora.com/Jeremy-McMinis) or [Github \(https://github.com/mcminis1\)](https://github.com/mcminis1).

Posted by Jeremy McMinis May 10, 2014


Comments

0 Comments

Jeremy's Blog

 Login ▾

 Recommend 8

 Share

Sort by Best ▾



Start the discussion...




LOG IN WITH

OR SIGN UP WITH DISQUS 



Name

Be the first to comment.

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Privacy

Recent Posts

- Support Vector Regression and Gaussian Process Example (<http://mcminis1.github.io/blog/2014/05/26/intuition-for-SVR-II/>)

- [Intuition for Support Vector Regression and the Gaussian Process \(http://mcminis1.github.io/blog/2014/05/10/intuition-for-SVR/\)](http://mcminis1.github.io/blog/2014/05/10/intuition-for-SVR/)
- [Building Better Products with Cohort Analysis & KPI Tracking \(http://mcminis1.github.io/blog/2014/04/27/getting-started/\)](http://mcminis1.github.io/blog/2014/04/27/getting-started/)

Categories

- [machine learning \(http://mcminis1.github.io/category/machine-learning.html\)](http://mcminis1.github.io/category/machine-learning.html)
- [product \(http://mcminis1.github.io/category/product.html\)](http://mcminis1.github.io/category/product.html)

Tags

Copyright © 2014 - Jeremy McMinis - Powered by Pelican (<http://getpelican.com>)