Exploring Classification

SUPPORT VECTOR MACHINE

We implemented the dual form of linear SVM's with slack variables in Python. To do so, we converted the input into a quadratic program, which we used `cvxopt.solvers.qp` to solve. Recall the general form of the quadratic program for the dual form of the SVM, which is given by

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)})$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

We will demonstrate the formulation of this program in the matrix form required for `cvxopt.solvers.qp` on a simple example. Consider the dataset with positive examples $(1, 2)$ and $(2, 2)$ and negative examples $(0, 0)$ and $(-2, 3)$. If we order the dataset according to the order presented here, then in matrix notation, the quadratic program above is given by

$$\text{minimize} \quad \frac{1}{2} \alpha^T \begin{bmatrix} 5 & 6 & 0 & -4 \\ 6 & 8 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -4 & -2 & 0 & 13 \end{bmatrix} \alpha + \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}^T \alpha$$

$$\text{subject to} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \alpha \leq \begin{bmatrix} C \\ 0 \\ C \\ 0 \\ C \\ 0 \\ C \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} \alpha = [0]$$

Solving this linear program for α, we obtain the linear separator shown in Figure 1. For this small example, we can clearly see that this separator maximizes the margin.

Applications

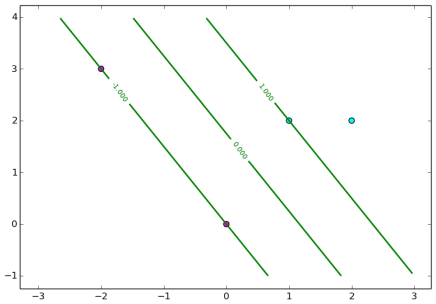


Figure 1: test

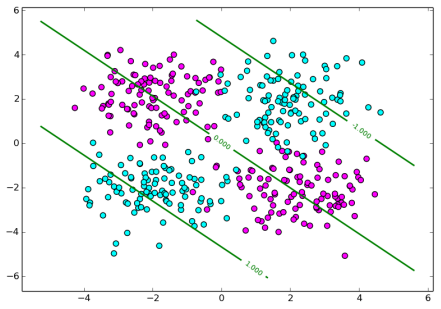
Next, we try out our implementation on four different data sets, each consisting of 400 points. The first three data sets, which we will refer to as `stdev1`, `stdev2`, `stdev4`, consist of points where each of the negative and positive examples are generated from a Gaussian distribution with varying standard deviations (as suggested by the name). The final data set, `nonSep`, is a data set generated as to be impossible to separate with a linear separator.

For each of these data sets we will use a value of $C = 1$, and we will show the resulting boundary on both a training and validation set. We will also report the training and validation error rates. We begin with `stdev1`.

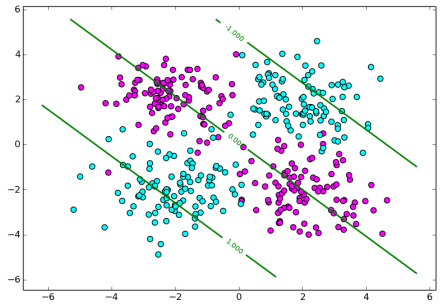
Figures 2(a) and 2(b) show the decision boundary formed by our algorithm on the training and validation data, respectively. As we can clearly see, both the training and validation data have an error rate of 0. This is not surprising given the very large gap between positive and negative samples.

Figure 2 also shows the results of repeating the process with `stdev2` and `stdev4`. For `stdev2`, the training error rate was 9.25% and the validation error rate was 8.25%. For `stdev4`, the training error rate was 26.5% and validation error rate 23.5%.

We can clearly see from the plots that `stdev2` and `stdev4` are not linearly separable (both in training and validation data), and thus we are not surprised by the increased error rates. It is interesting to note



(a) `nonSep` Training Data



(b) `nonSep` Validation Data

that the training data has higher error rates than the validation data in both cases. However, since both training data and validation data were drawn from Gaussian distributions, this is not terribly unlikely and does not surprise us.

Next, we focus on `nonSep`. The resulting decision boundaries are shown in Figure 3. Clearly, this data is ill-fitted for classification via a linear separator.

With a training error of 48.5%, we can see that the separator is barely able to do a better job than randomly guessing how to classify each point. On the validation data, we see an error rate of 50.75%. Because we have an error rate of over 50%, we could have done a better job if we guessed the exact opposite of what our trained separator suggested. If we were not sure form the plot of the data that this data was not well suited to be classified using a linear separator, then the very high error rates convince us that this is in fact the case.

Kernels

Next, we extended our dual form SVM to operate with kernels. We experimented on our data using the Gaussian kernel, with a variety of values for C and β. For each data set, we found the values of C and β which minimization the validation error rate.

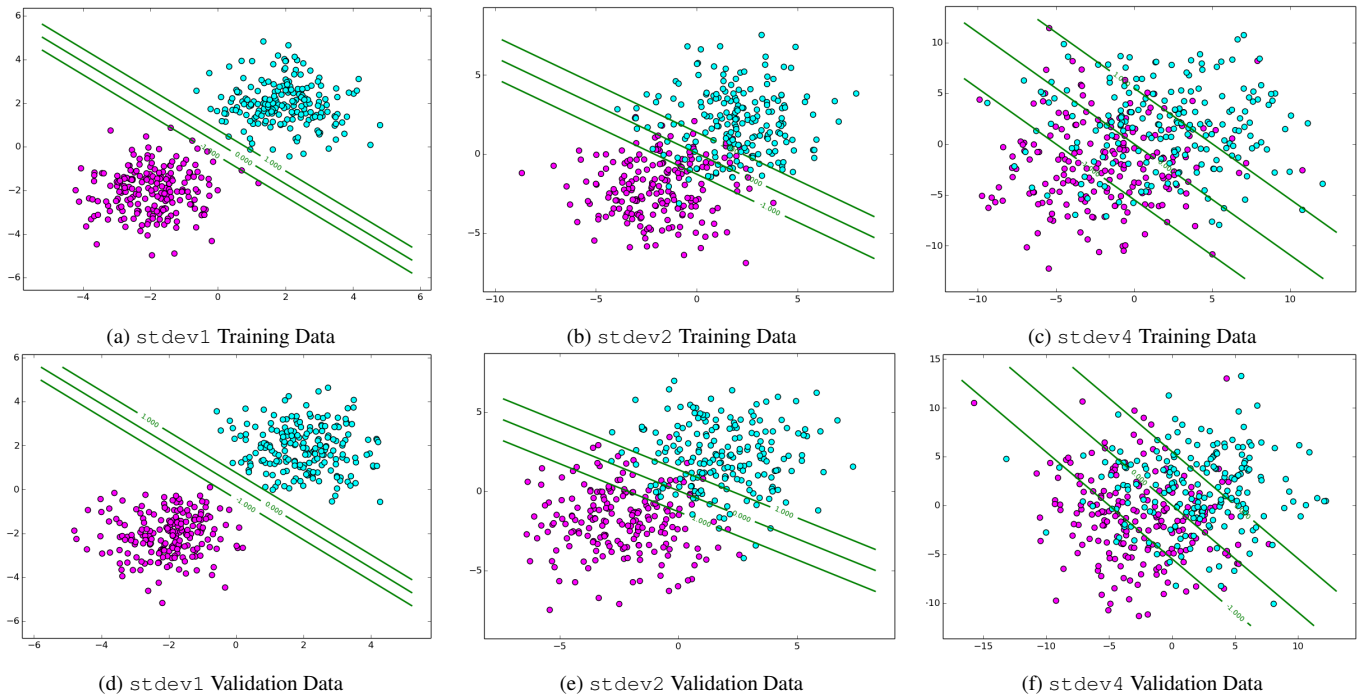
We repeated this experiment for the linear kernel, again using the validation error rate to tune C. The results are summarized in Figure 4.

For `stdev1`, `tdev2`, and `stdev4`, we can see that the test error rates resulting from both the Gaussian and linear kernels are very similar. From looking at the data sets, it seems that a linear separator is the most appropriate choice for the data. Thus, even when we apply a Gaussian separator, which allows a lot more flexibility, we still find that the linear separator performs roughly equally well.

For `nonSep`, we see a much different story. The data is clearly not linearly separable. Thus, we are hardly able to achieve an error rate below 50% with the linear separator. However, when we apply the Gaussian separator, we are able to achieve an error rate of only 5%. Thus, the added flexibility of the Gaussian kernel allows us to use the same techniques to separate data which is not linearly separable.

Figure 4 also shows the number of support vectors that the optimal parameters returned. We note that these numbers may be slightly misleading. Because `cvxopt.solvers.qp` never actually drove α values completely to zero, we had to choose a threshold such that we considered only α values above the threshold to be support vectors. For all of our experiments, we chose a value of 10^{-5}. From observation, this value seemed to most accurately represent a cutoff between α values which were significant and those which seemed as if they should be considered zero.

It is interesting to note that for all data sets other than `stdev1` (which was completely linearly separable and thus did not actually require any slack), the number of support vectors in the optimal solution was very high. It is possible that our threshold for deciding support vectors just did not do an adequate job of deciding which points should be considered support vectors. However, experimenting with the threshold value and the parameters of the solver did not yield any better results. We also note that for `stdev2` and `stdev4`, the value of C is very small. In the next section, we will discuss the effect that the value of C has on the number of



(a) `stdev1` Training Data (b) `stdev2` Training Data (c) `stdev4` Training Data

(d) `stdev1` Validation Data (e) `stdev2` Validation Data (f) `stdev4` Validation Data

Figure 2: test

	`stdev1`	`stdev2`	`stdev4`	`nonSep`
C	1	0.01	0.01	10
b	0.1	0.1	0.01	1
SVs	22	400	100	99
Test Error	0.25%	6%	21.75%	5%

(a) Gaussian Kernel

	`stdev1`	`stdev2`	`stdev4`	`nonSep`
C	10	0.01	0.01	1
SVs	3	125	249	392
Test Error	0	7.25%	22%	49.5%

(b) Linear Kernel

Figure 4: test

support vectors.

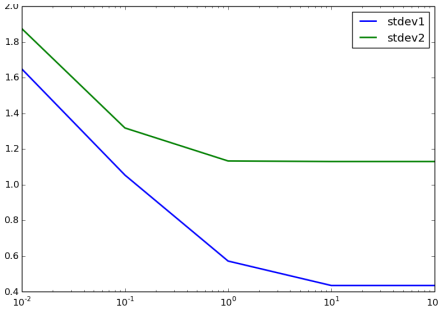
Analyzing C

Next, performed experiments to analyze the effects of varying C. Our goal is to examine how the value of C affects the value of the geometric margin and the number of support vectors chosen using the linear kernel. Note that the notion of geometric margin does not make sense for the Gaussian kernel. However, we note that the dependence of the number of support vectors on C was similar in the Gaussian kernel as in the linear kernel. We will show our results for data sets `stdev1` and `stdev2`. The results for the other two data sets are similar, so we do not display them here.

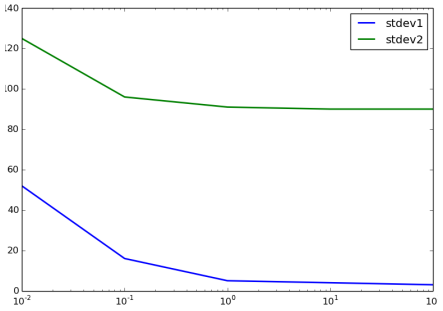
For each of `stdev1` and `stdev2`, we used C in the set $\{0.01, 0.1, 1, 10, 100\}$, and computed the geometric margin and total number of support vectors. As a general trend, both the geometric margin and the number of support vectors decreased as we increased C.

Figure 4 shows geometric margin and number of support vectors plotted as a function of C. Note that the x axis is shown on a log scale for convenience. We can see that both the geometric margin and number of support vectors seem to decrease to a point and then level off. This is because as we increase C, we decrease the amount of slack that we allow. As C approaches infinity, we approach the situation where we do not allow any slack.

We know that a support vector machine aims to



(a) Geometric Margin



(b) Number of Support Vectors

maximize the size of the margin. Thus, a seemingly reasonable idea would be to use C to maximize the value of the margin on the training data. However, from the trends we see above, this seems that it would just result in the choice of an arbitrarily small value for C, which we would have no reason to expect to perform well on the validation or test data. A more reasonable choice is to choose C based on the validation data. An option would be to minimize the size of the margin on the validation data. However, we found that the option which tended to give better overall performance was to choose C to minimize the validation error rate.