Author: Benjamin Smidt
Created: September 9, 2022
Last Updated: September 9, 2022

Assignment 2: Multiclass SVM

Loss

0.0.1 Resources

- CS 231N: SVM's and Softmax
- StatQuest with Josh Starmer

0.0.2 Mathematics

Let's begin by defining some variables. X is an NxD matrix containing N examples and D dimension for each example. W is a DxC matrix containing weights for a given example x_i in X where C indicates the number of classes we want to be able to classify x_i into. From here we can look at the loss function for Multiclass Support Vector Machines.

$$L_i = \sum_{j \neq y_i}^{C} max(0, s_j - s_{y_i} + \Delta) \tag{1}$$

$$s_j = f(x_i, W)_j = (Wx_i)_j \tag{2}$$

s_j is the "score" for class j of a single example x_i computed by taking the dot product of weight vector w_j (a column of W) and a single example x_i. s_{y_i} is the s_j value that is the correct classification for example x_i. L_i is the loss for a single example x_i and Δ is a fixed margin for which SVM "wants" the correct class score s_{y_i} to be greater than all the other class scores (s_j where $j \neq y_i$).

This loss is called *hinge loss* because the loss is 0 as long as $(s_{y_i} - s_j) \geq \Delta$. SVM doesn't care how large $s_{y_i} - s_j$ is, only that is at least as large as Δ. It could be infinite or equal to Δ, and it will treat those scores the same since they both have a loss of 0. You may also see the squared hinge loss

used, which is of the form $max(0, -)^2$ instead of $max(0, -)$. Finally, another (possibly helpful) way to write the loss function is:

$$L_i = \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \tag{3}$$

0.0.3 Programming

To compute the loss for an entire training set, we compute the average of the losses for every example in the training dataset. Computing the loss and the gradient using loops is fairly trivial, so I won't go into detail as to how to code this. However, it's a quick and useful implementation to see if your math is correct before going to the trouble (at least for me) of vectorizing your code. Note that the code contains regularization, which you can read more about below.

0.1 Regularization

Support Vector Machines have an important bug that is crucial to understand. Recall the loss function is defined as follows:

$$L_i = \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \tag{4}$$

The geometric interpretation of this loss function is that the SVM algorithm tries to find a single hyperplane (generalization of a dividing line for any number of dimensions) for each class that divides the correct classifications from every other classification for all examples in the dataset. If you haven't encountered this interpretation before, I highly suggest you watch this video. Many times this is impossible so the SVM algorithm opts for the hyperplane that simply minimizes the magnitude of misclassifications (which is the loss L_i).

It turns out that the weight vector w_j, which can be interpreted as the hyperplane dividing the correct classifications from all other classifications, is unique as long the data isn't linearly separable. Said another way, the hyperplane dividing correct and incorrect classifications is unique as long as

it is impossible to achieve 0 loss. As you may be able to guess, this is not the case when the dataset is linearly separable (0 loss can be achieved).

If 0 loss can be achieved, then we can scale w_j by any constant λ and still achieve 0 loss on the dataset. This is because scaling w_j, the hyperplane, doesn't change the hyperplane at all. Think of a line (a one dimensional hyperplane separating data in two dimensions). If we scale each dimension describing the line (in the case both the "rise" and the "run") by a constant, the slope doesn't change ($\frac{3}{4}x = \frac{6}{8}x$).

What does change for the SVM algorithm is the magnitude of the scores computed using those scaled weights. And since SVM's use hinge loss, if 0 loss can be achieved, then the SVM algorithm will arbitrarily scale w_j so that the margin Δ is as large as possible. This can result in an infinite feedback loop of scaling w_j (see CS 229 PS2.1)

To fix this issue we introduce regularization, a term we tack onto the end of the loss function that makes the loss function favor small weights. This term is known as the *regularization loss* while our original loss function terms are known as the *data loss*. Here's an example of L2 regularization which uses the L2 norm.

$$L = 1/N \sum_{i}^{N} \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) + \lambda \sum_{k}^{D} \sum_{l}^{C} W_{k,l}^2 \tag{5}$$

Regularization only makes sense in the context of a dataset, not a single example, so we use L instead of L_i. The first term, our data loss term, is simply indicates the average of all the losses L_i, so nothing new (just different notation). The second term, our regularization loss, is just the sum every squared value in the weight matrix multiplied by some constant λ. λ is there for us to be able to tell the SVM algorithm how important it is for it to keep the values in the weight matrix small.

Again, regularization just keeps the SVM algorithm from arbitrarily scaling linearly seperable datasets. It does have other nice properties and there are different types of regularization. For more details on this, see the References section.

0.2 References

1. CS 231N: SVM's and Softmax
2. StatQuest with Josh Starmer