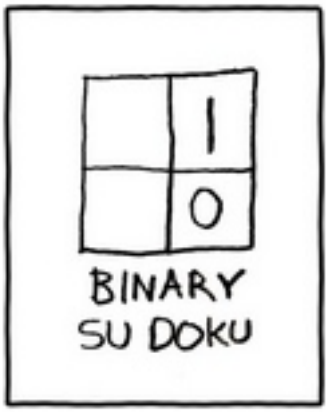
CS158 - Assignment 6
A Gradient Descent Into Madness

Due: Friday, October 14 by 6:00pm



For this assignment we're going to try out a few gradient descent variants (including SVM). The amount of code you'll need to write for this will be pretty minimal.

1 High-level requirements

- Implement the gradient descent algorithm
- Your approach will need to support two loss functions: hinge loss and exponential loss
- Your approach will need to support three regularization type: none, L1 and L2
- You will provide a short writeup that includes:
  - an argument that your implementation is correct. This can be of the form of examining text snippets with justification and/or showing experimental results.
  - at least one experimental result

2 Implementation Requirements

To get you started I have included some starter code at:

`http://www.cs.pomona.edu/~dkauchak/classes/cs158/assignments/assign6/assign6-starter.tar.gz`

This is more or less the same starter code as assignment 5 except I have add the `KNNClassifier` (so that everything compiles) and I have included a starter class `GradientDescentClassifier`. This file is a copy of the `PerceptronClassifier` file with the addition of a few constants. This should serve as your starting point.

Implement a gradient descent classifier. Your implementation must:

- support a zero parameter constructor. By default you should use exponential loss, no regularization, $\lambda = 0.1$ and $\eta = 0.1$
- include a method `setLoss` that takes an int and selects the loss function to use (based on the constants)
- include a method `setRegularization` that takes an int and selects the regularization method to use (based on the constants)
- include a method `setLambda` that takes a double and sets the lambda to use
- include a method `setEta` that takes a double and sets the eta to use (Note, in practice we'd use some sort of schedule of etas, e.g. one for each iteration, however, to keep it simple we'll just have a constant learning rate.)
- support training and testing based on these parameters
- update the model weights (both w_i and b) after *every* example. There are two common ways for implementing gradient descent:

  1. For each iteration, iterate over all the examples and aggregate the sum of the adjustments and then adjust the weights based on this. This results in adjusting each of the weights of the model once *per iteration over the entire data*.
  2. Or, for each iteration, for each example, calculate the adjustment to the weights and then immediately adjust the model weights before moving to the next example.

  Both are correct and are gradient descent approaches in that they will make steps towards lower gradient. We're going to implement option 2 for this assignment. In practice, it works better to update every example, particularly for larger data sets. The trajectory is a bit noisier since it's only based on a single example, however, the advantage is that you take into account changes in the weights immediately into the model for future gradient calculations.

3 Writeup

In addition to your code, include a writeup (in some reasonable file format) that includes two sections: **Algorithm Correctness** and **Experimentation**. Because of the small amount of coding required for this assignment, the writeup will be a non-trivial part of your grade, so make sure to devote sufficient time to this.

Algorithm Correctness

One of the challenges with implementing machine learning algorithms, particularly iterative approaches like this, is determining if they are operating correctly. Include a short (∼1 page), concrete justification of why your implementation of the hinge loss with L2 regularization is correct. This could include snippets of code, along with an explanation, and/or short experimental results, e.g. incremental output of a small problem. *You will be graded based on how convincing and thorough your argument is.*

Experimentation

Run one experiment that highlights something interesting about the algorithm, include the resulting data, and include a few sentences of explanation/analysis. For example, you could investigate the optimal λ/η for one of the variants on one of the data sets, or you could investigate how the different loss/regularization approaches work (though pick some reasonable lambda). Plan on spending around an hour playing around with this. *You will be graded based on the quality of your experimental setup and your presentation.*

4 Hints/observations

- Tuning the hyperparameters (i.e. λ and η) is extremely important for gradient descent to get the algorithms to work well.
- Think about how you're going to tune the parameters. When you have multiple hyperparameters the space of possible parameters goes up drastically since you have to consider possible combinations of both. There are many ways to do this (be creative), but as before, definitely do it programmatically.
- Exponential loss is very hard to tune for as the accuracy tends to jump around a lot. Start with the hinge loss first and then start at similar values for the exponential loss.
- You can use whatever dataset you'd like, however, datasets with a large number of features (and, therefore, a larger number of weights and higher dimensionality), and particularly sparse datasets (like our wine dataset), are going to be harder/slower to train for iterative methods like gradient descent.
- Don't forget that b is also a parameter of the model and needs to be updated each time. One way that is helpful to think about b is to think of it like any of the other weights that happens to be associated with a feature whose value is always 1. You should update it whenever and in the same way as the other weights.
- In most places in the gradient descent notes y' is shorthand for $wx+b$, not for the classification prediction (i.e. +1 or -1). If you look at the loss functions, they do want to take into account not only the classification prediction, but also how far away from the hyperplane the prediction is.

5 Extra Credit

For those who would like to experiment (and push themselves) a bit more (and of course, get a bit of extra credit) you can try out some of these extra credit options. If you try out these options, include an extra file called `extra.txt` that describes what extra credit you did.

- Allow for an η schedule rather than just a single constant. I'll let you be creative about the best way to do this, but please just create a new copy of your classifier code rather than trying to alter the existing code.
- Add additional regularization or loss functions. If you do this, add appropriate constants for selecting these.
- Do additional experimentation and include it in the write-up. The amount earned will be based on how much effort you put in and how creative the experiment is.

6 When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Create a directory with your last name, followed by the assignment number, for example, for this assignment mine would be `kauchak6`. If you worked with a partner, put both last names.

Inside this directory, create a `code` directory and copy all of your code into this directory, maintaining the package structure.

Finally, also include your `writeup` file.

`tar` then `gzip` this folder and submit that file on the submission page on the course web page.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate JavaDoc
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.