Exercises 3: Better online learning (preliminaries)

The goal of the next two sets of exercises is to make your SGD implementation better, faster, and able to exploit sparsity in the features. These exercises set the stage. On the next set, you'll then put everything together.

Once again, we'll return to the logistic-regression model, by now an old friend: $y_i \sim \text{Binomial}(m_i, w_i)$, where y_i in an integer number of "successes," m_i is the number of trials for the ith case, and the success probability w_i is a regression on a feature vector x_i given by the inverse logit transform:

$$w_i = \frac{1}{1 + \exp\{-x_i^T \beta\}} .$$

As before, we'll use $l(\beta)$ to denote the loss function to be minimized: that is, the negative log likelihood for this model.

Before we get to work on improving stochastic gradient descent itself, we need to revisit our batch[1] optimizers from the first set exercises: ordinary gradient descent and Newton's method.

1 Line search

Line search is a technique for getting a good step size in optimization. You have may already implemented line search on the first set of exercises, but if you haven't, now's the time.

Our iterative (batch) algorithms from the previous exercises involved updates that looked like this:

$$\beta^{(t+1)} = \beta^{(t)} + \gamma s^{(t)} ,$$

where $s^{(t)}$ is called the search direction. We tried two different search directions: the gradient-descent direction (i.e. in the opposite direction from the gradient at the current iterate), and the Newton direction.

In either case, we have the same question: how should we choose γ? It's clear that the best we can do, in a local sense, is to choose γ to minimize the one-dimensional function

$$\phi(\gamma) = l(\beta^{(t)} + \gamma s^{(t)}) ,$$

There's no other choice of γ that will lead to a bigger decrease in the loss function along the fixed search direction $s^{(t)}$. While in general it might be expensive to find the exact minimizing γ, we can do better than just guessing. Line search entails using some reasonably fast heuristic for getting a decent γ.

Here we'll focus on the gradient-descent direction. Read Nocedal and Wright, Section 3.1. Then:

(A) Summarize your understanding of the *backtracking line search* algorithm based on the Wolfe conditions (i.e. provide pseudo-code and any necessary explanations) for choosing the step size.

Based on my understanding, Wolfe conditions are a set of inequalities for performing inexact line search. The purpose of inexact line searches with Wolfe conditions is to provide an efficient way of computing an "acceptable" step length that reducing the objective function in each iteration. In other words, we would like to select a step size to minimize the function $f(x_k + ap_k)$

(1) The first Wolfe condition is the Armijo condition:

$$(x_k + a_k p_k) \leq f(x_k) + c1 a_k p k^T \nabla\_k^T$$

This sometimes called the sufficient decrease condition, but did not guarantee the stepsize is meaningful. To obtain the reasonable size, we also need the second Wolfe conditions of "curvature condition"

(2) Curvature condition:

[1] Batch, in the sense that they work with the entire batch of data at once. The key distinction here is between batch learning and online learning, which processes the data points either one at a time, or in mini-batches.