ODS-24 sratch

Tanner Aaron Graves

May 2024

1 Introduction

In this report we will analyze the relative performance of various gradient methods at optimizing a multi-class logistic regression model. Specifically, we are interested in comparing full gradient descent with block coordinate based methods (BCGD) which optimize one coordinate, or column in parameter matrix X, at a time. Multi-class logistic classification is an extension of traditional logistic regression. The task goes from predicting a Bernoulli random variable to predicting one label to classify each example from a set of k mutually exclusive labels. We encode the target $b_i \in \mathbb{R}^k$ where k is the number of output classes to be a one hot vector, corresponding to the correct classification of example $a_i \in \mathbb{R}^d$. It assigns these labels by learning the conditional probability distribution that the correct label is b_i given example a_i and learned parameter matrix $X \in \mathbb{R}^{d\times k}$ Where d is the number of features in each example. The output distribution is computed with the softmax function in this context:

$$p(b_i|a_i, X) = \frac{\exp(x_{b_i}^T a_i)}{\sum_{c=1}^k \exp(x_c^T a_i)}$$

Multi-class logistic regression is a simple yet effective method with ubiquitous use. Its training does, however, require learning the parameter matrix X, which can be done using various methods like maximum likelihood or gradient descent based methods. This corresponds to solving the following optimization problem:

$$\min_{x\in\mathbb{R}^{d\times k}} \sum_{i=1}^m [-x_{b_i}^T a_i + \log(\sum_{c=1}^k \exp(x_c^T a_i))]$$

which is the cross-entropy of the predicted class distribution and the target label b_i. Of importance to the application of gradient methods is this problems convexity. This is nice for many reasons, but most importantly we can guarantee that a local minima found by GD methods will be the global minimum.

2 Algorithms

Here we provide an overview to the methods studied. We implemented the various methods in python using either pyTorch or Numpy. The three algorithms studied are full gradient descent and two BCGD methods using both Gauss-Southwell and randomized rule.

2.1 Gradient Descent

Algorithm 1 Full Gradient Descent

```
Initialize $X_1 \in \mathbb{R}^{d \times k}$
**for** $k = 1, ...$ **do**
    If $X \in X^*$ **STOP** (Optimality conditions)
    Set $X_{k+1} = x_k - \alpha_k \nabla$
**end for**
```

2.2 Block coordinate Gradient Descent

BCGD methods are specialized in optimizing high dimensional problems where where calculation of the gradient can be split up in to blocks which are easier to compute. These methods exploit the convex nature of the problem to optimize one coordinate at a time and approach the global optimum. Above $\alpha_k \geq 0$ is

Algorithm 2 BCGD

```
Initialize $X_1 \in \mathbb{R}^{d \times k}$
**for** $k = 1, ...$ **do**
    If $X \in X^*$ **STOP** (Optimality conditions)
    Select $i_k \in$ block according to some **RULE**
    Set $x_{k+1} = x_k - \alpha_k [\nabla_x f(x_k)]_{i_k}$
**end for**
```

a stepsize for the kth iteration. For most of our analysis we fix $\alpha_k = \frac{1}{L}$, but experiment with the effect of different choices in the Stepsize Analysis subsection of the Synthetic Dataset section. The choice of **RULE** will distinguish the Gauss-Southwell and Randomized variants of BCGD.

3 Synthetic Dataset

3.1 Creation

3.2 Performance

3.3 Stepsize Analysis

4 Real-world Dataset