Section 06: Classification, Gradient Descent Review

1. Binary Logistic Regression

In classification problems, we predict a discrete label y given features x. The standard approach is to model the conditional probability $P(Y = y \mid x)$ and then predict the most likely class. For binary classification, **logistic regression** models this probability for binary labels $y \in \{0, 1\}$ as:

$$P(Y = 1|x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

and

$$P(Y = 0|x) = 1 - \sigma(w^T x) = 1 - \frac{1}{1 + e^{-w^T x}} = \frac{e^{-w^T x}}{1 + e^{-w^T x}}$$

where $w \in \mathbb{R}^d$ is the weight vector.

(a) As we saw in lecture, it is often easier to use labels $y \in \{-1, +1\}$ for optimization. Using this technique, show that $P_w(y|x) = \frac{1}{1+\exp(-yw^T x)}$ correctly represents both cases from above (i.e., $P(Y = 1|x)$ and $P(Y = 0|x)$).

(b) Given a dataset of n i.i.d. observations $\{(x_i, y_i)\}_{i=1}^n$ where $y_i \in \{-1, +1\}$, use maximum likelihood estimation to derive the formula for logistic loss, which is defined as:

$$\hat{w}_{MLE} = \arg\min_w \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

(c) Suppose we have a dataset that is linearly separable. If we use the objective function derived in part (b) and run gradient descent, will it converge to a specific, finite weight vector w? What happens to $||w||$?

2. Multi-class Logistic Regression

When we move beyond binary classification to $k > 2$ classes, we use the **softmax function** to represent the conditional probability $P(Y = c_j|x)$ for each class c_j:

$$P(Y = c_j|x) = \frac{e^{w_j^T x}}{\sum_{j'=1}^{k} e^{w_{j'}^T x}}$$

The probability that a given input x belongs to class c_j is proportional to the exponential of its logit $w_j^T x$, normalized by the logits of all other possible classes. This ensures that the predicted probabilities are all non-negative and sum to 1 across all k classes.

(a) Show that when $k = 2$, this reduces to the binary logistic regression model. Specifically, treat class c_1 as $y = 1$ and show that $P(Y = c_1|x) = \sigma(w^T x)$ for some weight vector w. What is w in terms of w_1 and w_2?

(b) Suppose a 3-class classifier produces logits $w_1^T x = \log 3$, $w_2^T x = \log 5$, and $w_3^T x = \log 1$ on some input x. Compute the softmax probabilities. Which class is predicted?

3. Stochastic Gradient Descent

Consider minimizing an average of functions:

$$\min_w \frac{1}{n}\sum_{i=1}^{n}\ell_i(w),$$

where w is a $d-$dimensional vector (or the feature dimension is d). The minimization of the negative of a log-likelihood function can serve as an example. Recall that the (full) gradient descent step is given by

$$w^{(t+1)} = w^{(t)} - \eta \cdot \frac{1}{n}\sum_{i=1}^{n}\nabla\ell_i\big(w^{(t)}\big).$$

The computational cost of a single step here is $\mathcal{O}(dn)$. To reduce cost, one idea is to just use a subset of all samples to approximate the full gradient. Specifically, consider revising the gradient descent step as follows:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla\ell_{I_t}(w^{(t)}),$$

where I_t is chosen randomly within $\{1, 2, ..., n\}$ with equal probability. This is called **stochastic gradient descent (SGD)**, and the computational cost of a single step now reduces to $\mathcal{O}(d)$.

(a) What disadvantages can SGD have? How can we balance between the noise in updates and computational cost?

4. Extensions of SGD

(a) Gradient descent requires the full gradient when updating while (standard) SGD utilizes the gradient of one sample when updating. **Mini-batching** is somewhere between the two extremes. That is, we choose a random subset $I_t \subseteq \{1, ..., n\}$ with size $|I_t| = b \ll n$ in the stochastic gradient descent step:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \frac{1}{b}\sum_{i_t\in I_t}\nabla\ell_{i_t}(w^{(t)}).$$

With mini-batching, we have the following results:

- $\mathbb{E}_{I_t}\big[\frac{1}{b}\sum_{i_t\in I_t}\nabla\ell_{i_t}(w^{(t)})\big] = \frac{1}{n}\sum_{i=1}^{n}\nabla\ell_i(w^{(t)})$: we still have an unbiased estimate of the full gradient.
- Compared to standard SGD, variance of the gradient estimate is reduced approximately by $\frac{1}{b}$.
- Computational cost for each step now becomes $\mathcal{O}(db)$.

Remark: By matrix computations (computing b gradients at a time) and parallelization, we can denoise the estimated gradients without increasing much computational cost (for batch size b that is not large).

(b) How should we choose the batch size?

(c) Are there other extensions or variants of the basic stochastic gradient descent algorithm?