Section 06: Solutions

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 $P(Y = 0|x)$).
Solution:

First, we swap out the labels:

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

Then, note that:

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

In other words, we've shown the useful identity $1 - \sigma(z) = \sigma(-z)$.

Now we can verify the unified form $P_w(y|x) = \frac{1}{1+\exp(-yw^T x)}$ for both cases:

- For $y = +1$: $P_w(+1|x) = \frac{1}{1 + e^{(-yw^T x)}} = \frac{1}{1 + e^{(-(+1)w^T x)}} = \frac{1}{1 + e^{-w^T x}} = P(Y = +1|x)$
- For $y = -1$: $P_w(-1|x) = \frac{1}{1 + e^{(-yw^T x)}} = \frac{1}{1 + e^{(-(-1)w^T x)}} = \frac{1}{1 + e^{w^T x}} = P(Y = -1|x)$

Both match the original expressions, thus $P_w(y|x) = \frac{1}{1 + e^{(-yw^T x)}}$ correctly represents both cases.

(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:

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

Solution:

We have n i.i.d. observations $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$. Then:

$$P_w(y_i|x_i) = \sigma(y_i w^T x_i) = \frac{1}{1 + \exp(-y_i w^T x_i)}$$

$$\downarrow$$

$$P_w(y|x) = \prod_{i=1}^{n} \sigma(y_i w^T x_i) = \prod_{i=1}^{n} \frac{1}{1 + \exp(-y_i w^T x_i)}$$

$$\downarrow$$

$$\ell_w = \log P_w(y|x) = \sum_{i=1}^{n} \log(\sigma(y_i w^T x_i)) = \sum_{i=1}^{n} \log\left(\frac{1}{1 + \exp(-y_i w^T x_i)}\right)$$

$$= -\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) \qquad \text{[Note: } \log(\tfrac{1}{a}) = \log(a^{-1}) = -\log(a)\text{]}$$

$$\downarrow$$

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

(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 $||w||$?
Solution:

No; it will not converge to a specific, finite weight vector. Because the data is linearly separable, there exists some w^* such that $y_i(w^*)^T x_i > 0$ for all i, and scaling w^* by any $c > 0$ still separates the data. And due to the exponential term in the logistic loss function, the larger the weights get, the closer the loss will be to 0 (i.e., as $w \to \infty$, $\exp(-y_i w^T x_i) \to 0$ and thus $\log(1 + \exp(-y_i w^T x_i)) \to 0$). As we take more and more steps, $||w|| \to \infty$.

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?
Solution:

Plugging in $k = 2$ into the equation, we get:

$$P(Y = c_1|x) = \frac{e^{w_1^T x}}{e^{w_1^T x} + e^{w_2^T x}} \cdot \frac{e^{-w_1^T x}}{e^{-w_1^T x}} = \frac{1}{1 + (e^{w_2^T x} \cdot e^{-w_1^T x})} = \frac{1}{1 + e^{(w_2 - w_1)^T x}} = \frac{1}{1 + e^{-(w_1 - w_2)^T x}}$$

Let $w = w_1 - w_2$. We get:

$$\frac{1}{1 + e^{-(w_1 - w_2)^T x}} = \frac{1}{1 + e^{-w^T x}} = \sigma(w^T x)$$

Therefore, $w = w_1 - w_2$.
This shows that softmax with $k = 2$ is equivalent to binary logistic regression; they're the same model, just written with different parameterizations.

(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?
Solution:

We plug in the given values into the softmax function for each class:

$$P(Y = c_1|x) = \frac{e^{w_1^T x}}{\sum_{j'=1}^{3} e^{w_{j'}^T x}} = \frac{e^{\log 3}}{e^{\log 3} + e^{\log 5} + e^{\log 1}} = \frac{3}{3 + 5 + 1} = \frac{1}{3}$$

$$P(Y = c_2|x) = \frac{e^{w_2^T x}}{\sum_{j'=1}^{3} e^{w_{j'}^T x}} = \frac{e^{\log 5}}{e^{\log 3} + e^{\log 5} + e^{\log 1}} = \frac{5}{3 + 5 + 1} = \frac{5}{9}$$

$$P(Y = c_3|x) = \frac{e^{w_3^T x}}{\sum_{j'=1}^{3} e^{w_{j'}^T x}} = \frac{e^{\log 1}}{e^{\log 3} + e^{\log 5} + e^{\log 1}} = \frac{1}{3 + 5 + 1} = \frac{1}{9}$$

Class 2 is predicted, since it has the highest probability.

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?

Solution:

By treating SGD as noise-injected gradient descent:

$$\nabla\ell_{I_t}(w^{(t)}) = \mathbb{E}_{I_t}\Big[\nabla\ell_{I_t}(w^{(t)}\Big] + e_t = \frac{1}{n}\sum_{i=1}^{n}\ell_i(w^{(t)}) + e_t,$$

where e_t represents the noise term and is random, we know that the steps taken towards a minimum can be very noisy because the gradient used in updating involves noise. One way to balance the noise in updates and computational cost is to consider a technique called **mini-batching**, which is employed with SGD.

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? **Solution:**

The choice of the optimal batch size is not an easy question, and there is no standard answer to it. However, we still try to provide some important intuitions regarding the choice of batch size. Firstly, when the objective function (to be minimized) behaves "better" (e.g., Lipschitz continuous, strong convex) than convex functions, the difference in the convergence rates between GD and SGD becomes significant, suggesting a nontrivial gain of having a faster convergence rate and hence we should consider relatively larger batch size. Secondly, a smaller batch size yields less stable gradient estimates, suggesting that we shall employ a fairly small step size/learning rate. An increase in the batch size can be paired with an increase in the step size/learning rate.

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

Solution:

Many improvements, which are listed below, on the basic SGD algorithm have been developed and used.

- Implicit updates (ISGD)
- Momentum
- Averaged stochastic gradient descent
- Adaptive gradient algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

Basically, these methods consider to fine-tune the step size parameter, take previous update magnitude into account, or introduce the second moments of the gradients when updating. For example, Momentum remembers the previous update magnitude so that $w^{(t)}$ tends to keep traveling in the same direction, preventing oscillations:

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

Adam, as another example, considers to tune to step size with the second moments of the gradients:

$$w^{(t+1)} = w^{(t)} - \eta G\Big(\nabla \ell^{(t)}, \nabla \ell^{(t-1)}, ..., (\nabla \ell^{(t)})^2, (\nabla \ell^{(t-1)})^2, ...\Big),$$

where $\nabla \ell^{(t)} = \nabla \ell_{I_t}(w^{(t)})$ and G is a function that involves element-wise square of all previous gradients. The paper below provides more details on Adam: https://arxiv.org/pdf/1412.6980.pdf.