Week of March 4, 2019

CPSC 340 Tutorial

Si Yi (Cathy) Meng *mengxixi@cs.ubc.ca*

1 Multi-class Logistic Regression

We want to learn the weights W (d by k) such that for each row in the output of

$$XW = Y$$

the value is highest at the index of the true label. As an example, consider the dataset where $d = 3$ and $k = 4$ with features being binary for simplicity. Suppose we fit a linear classifier using the softmax loss and obtain the following weights,

$$W = \begin{bmatrix} 1 & -5 & 4 & 1 \\ 2 & 2 & -1 & 0 \\ 2 & 3 & 0 & 1 \end{bmatrix}$$

.

Which class label would we assign to the following test example?

$$\hat{X} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

Solution:

$$\hat{Y} = \hat{X}W = \begin{bmatrix} 3 & -2 & 4 & 2 \end{bmatrix}$$

And therefore we predict class corresponds to index 2.

Let's take a closer look at the objective function using the softmax loss.

$$f(W) = \sum_{i=1}^{n} \Big[- w_{y_i}^T x_i + \log \Big(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \Big) \Big]$$

Since W is a d by k matrix, then we can use the class indices 1 through k to obtain the length d weight vector corresponding to that class. When we want to minimze this with gradient descent, we need find the gradient $\nabla f(W)$, which will have the same shape as W. Specifically, the partial derivative with respect to an element in W would be

$$\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij} [p(y_i = c \,|\, W, x_i) - I(y_i = c)]$$

where $p(y_i = c \,|\, W, x_i)$ is the probability of example i being class c, defined as

$$p(y_i = c \,|\, W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)}$$

To show the partial derivatives are indeed the above, it's easier if you just consider the loss for one training example, take the derivative with respect to a particular c and j. Some terms will disappear as they do not involve the w_c that we care about.

2 Kernel Trick

(lecture slides)

3 Stochastic Gradient Descent

Recall the gradient descent update step is:

$$w^{t+1} = w^t - \alpha^t \nabla f(w^t)$$

For least squares, the gradient is

$$\nabla f(w) = X^T(Xw - y) = \sum_{i=1}^{n} (w^T x_i - y_i) x_i$$

for a particular w. The cost for computing this gradient would be $O(nd)$, which is also the cost of an update. This becomes very expensive for a large dataset. Instead of summing over all n examples, we can instead randomly pick an example and evaluate the gradient

$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t)$$

where $\nabla f_i(w) = (w^T x_i - y_i) x_i$. Now our update cost of one iteration becomes $O(d)$, which no longer depend on n.

Pseudo-code for minibatch SGD:

```
init w = w0
define alpha, batch_size, n_epochs

for epoch in 1-n_epochs:
    X, y = shuffle(X, y)
    for i in range(0, N, batch_size):
        # N is the number of training examples

        grab the corresponding Xbatch, ybatch from shuffled X, y
        evaluate f, g using w, Xbatch and ybatch
        update w
```