Softmax Regression

Introduction

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression we assumed that the labels were binary: $y^{(i)} \in \{0,1\}$. We used such a classifier to distinguish between two kinds of hand-written digits. Softmax regression allows us to handle $y^{(i)} \in \{1,\ldots,K\}$ where K is the number of classes.

Recall that in logistic regression, we had a training set $\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$ of m labeled examples, where the input features are $x^{(i)} \in \Re^n$. With logistic regression, we were in the binary classification setting, so the labels were $y^{(i)} \in \{0,1\}$. Our hypothesis took the form:

$$h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)},$$

and the model parameters θ were trained to minimize the cost function

$$J(\theta) = -\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

In the softmax regression setting, we are interested in multi-class classification (as opposed to only binary classification), and so the label y can take on K different values, rather than only two. Thus, in our training set $\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$, we now have that $y^{(i)} \in \{1,2,\ldots,K\}$. (Note that our convention will be to index the classes starting from 1, rather than from 0.) For example, in the MNIST digit recognition task, we would have $K=10$ different classes.

Given a test input x, we want our hypothesis to estimate the probability that $P(y=k|x)$ for each value of $k=1,\ldots,K$. I.e., we want to estimate the probability of the class label taking on each of the K different possible values. Thus, our hypothesis will output a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities. Concretely, our hypothesis $h_\theta(x)$ takes the form:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

Here $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \Re^n$ are the parameters of our model. Notice that the term $\frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)}$ normalizes the distribution, so that it sums to one.

For convenience, we will also write θ to denote all the parameters of our model. When you implement softmax regression, it is usually convenient to represent θ as a n-by-K matrix obtained by concatenating $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ into columns, so that

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(K)} \\ | & | & | & | \end{bmatrix}.$$

Cost Function

We now describe the cost function that we'll use for softmax regression. In the equation below, $1\{\cdot\}$ is the "indicator function," so that $1\{\text{a true statement}\}=1$, and $1\{\text{a false statement}\}=0$. For example, $1\{2+2=4\}$ evaluates to 1; whereas $1\{1+1=5\}$ evaluates to 0. Our cost function will be:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)}=k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

Notice that this generalizes the logistic regression cost function, which could also have been written:

$$\begin{aligned} J(\theta) &= -\left[\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_\theta(x^{(i)})) + y^{(i)}\log h_\theta(x^{(i)})\right] \\ &= -\left[\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)}=k\right\}\log P(y^{(i)}=k|x^{(i)};\theta)\right] \end{aligned}$$

The softmax cost function is similar, except that we now sum over the K different possible values of the class label. Note also that in softmax regression, we have that

$$P(y^{(i)}=k|x^{(i)};\theta) = \frac{\exp(\theta^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})}$$

.

We cannot solve for the minimum of $J(\theta)$ analytically, and thus as usual we'll resort to an iterative optimization algorithm. Taking derivatives, one can show that the gradient is:

$$\nabla_{\theta^{(k)}}J(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)}=k\} - P(y^{(i)}=k|x^{(i)};\theta)\right)\right]$$

Recall the meaning of the "$\nabla_{\theta^{(k)}}$" notation. In particular, $\nabla_{\theta^{(k)}}J(\theta)$ is itself a vector, so that its j-th element is $\frac{\partial J(\theta)}{\partial \theta_{lk}}$ the partial derivative of $J(\theta)$ with respect to the j-th element of $\theta^{(k)}$.

Armed with this formula for the derivative, one can then plug it into a standard optimization package and have it minimize $J(\theta)$.

Properties of softmax regression parameterization

Softmax regression has an unusual property that it has a "redundant" set of parameters. To explain what this means, suppose we take each of our parameter vectors $\theta^{(j)}$, and subtract some fixed vector ψ from it, so that every $\theta^{(j)}$ is now replaced with $\theta^{(j)} - \psi$ (for every $j = 1, \ldots, k$). Our hypothesis now estimates the class label probabilities as

$$\begin{aligned} P(y^{(i)}=k|x^{(i)};\theta) &= \frac{\exp((\theta^{(k)}-\psi)^\top x^{(i)})}{\sum_{j=1}^{K}\exp((\theta^{(j)}-\psi)^\top x^{(i)})} \\ &= \frac{\exp(\theta^{(k)\top}x^{(i)})\exp(-\psi^\top x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})\exp(-\psi^\top x^{(i)})} \\ &= \frac{\exp(\theta^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x^{(i)})}. \end{aligned}$$

In other words, subtracting ψ from every $\theta^{(j)}$ does not affect our hypothesis' predictions at all! This shows that softmax regression's parameters are "redundant." More formally, we say that our softmax model is "'overparameterized,'" meaning that for any hypothesis we might fit to the data, there are multiple parameter settings that give rise to exactly the same hypothesis function h_θ mapping from inputs x to the predictions.

Further, if the cost function $J(\theta)$ is minimized by some setting of the parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(k)})$, then it is also minimized by $(\theta^{(1)} - \psi, \theta^{(2)} - \psi, \ldots, \theta^{(k)} - \psi)$ for any value of ψ. Thus, the minimizer of $J(\theta)$ is not unique. (Interestingly, $J(\theta)$ is still convex, and thus gradient descent will not run into local optima problems. But the Hessian is singular/non-invertible, which causes a straightforward implementation of Newton's method to run into numerical problems.)

Notice also that by setting $\psi = \theta^{(K)}$, one can always replace $\theta^{(K)}$ with $\theta^{(K)} - \psi = \vec{0}$ (the vector of all 0's), without affecting the hypothesis. Thus, one could "eliminate" the vector of parameters $\theta^{(K)}$ (or any other $\theta^{(k)}$, for any single value of k), without harming the representational power of our hypothesis. Indeed, rather than optimizing over the $K \cdot n$ parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)})$ (where $\theta^{(k)} \in \Re^n$), one can instead set $\theta^{(K)} = \vec{0}$ and optimize only with respect to the $K \cdot n$ remaining parameters.

Relationship to Logistic Regression

In the special case where $K = 2$, one can show that softmax regression reduces to logistic regression. This shows that softmax regression is a generalization of logistic regression. Concretely, when $K = 2$, the softmax regression hypothesis outputs

$$h_\theta(x) = \frac{1}{\exp(\theta^{(1)\top} x) + \exp(\theta^{(2)\top} x^{(i)})} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \end{bmatrix}$$

Taking advantage of the fact that this hypothesis is overparameterized and setting $\psi = \theta^{(2)}$, we can subtract $\theta^{(2)}$ from each of the two parameters, giving us

$$\begin{aligned} h(x) &= \frac{1}{\exp((\theta^{(1)} - \theta^{(2)})^\top x^{(i)}) + \exp(\vec{0}^\top x)} \left[\exp((\theta^{(1)} - \theta^{(2)})^\top x) \exp(\vec{0}^\top x) \right] \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ 1 - \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \end{aligned}$$

Thus, replacing $\theta^{(2)} - \theta^{(1)}$ with a single parameter vector θ', we find that softmax regression predicts the probability of one of the classes as $\frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, and that of the other class as $1 - \frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, same as logistic regression.

Exercise 1C

Starter code for this exercise is included in the Starter code GitHub Repo (https://github.com/amaas/stanford_dl_ex) in the `ex1/` directory.

In this exercise you will train a classifier to handle all 10 digits in the MNIST dataset. The code is very similar to that used for Exercise 1B except that it will load the entire MNIST train and test sets (instead of just the 0 and 1 digits), and the labels $y^{(i)}$ have 1 added to them so that $y^{(i)} \in \{1, \ldots, 10\}$. (The change in the labels allows you to use $y^{(i)}$ as an index into a matrix.)

The code performs the same operations as in Exercise 1B: it loads the train and test data, adding an intercept term, then calls `minFunc` with the `softmax_regression_vec.m` file as the objective function. When training is complete, it will print out training and testing accuracies for the 10-class digit recognition problem.

Your task is to implement the `softmax_regression_vec.m` file to compute the softmax objective function $J(\theta; X, y)$ and store it in the variable f. You must also compute the gradient $\nabla_\theta J(\theta; X, y)$ and store it in the variable g. Don't forget that `minFunc` supplies the parameters θ as a vector. The starter code will reshape θ into a n-by-(K-1) matrix (for K=10 classes). You also need to remember to reshape the returned gradient g back into a vector using $g = g(:)$;

You can start out with a for-loop version of the code if necessary to get the gradient right. (Be sure to use the gradient check debugging strategy covered earlier!) However, you might find that this implementation is too slow to run the optimizer all the way through. After you get the gradient right with a slow version of the code, try to vectorize your code as well as possible before running the full experiment.

Here are a few MATLAB tips that you might find useful for implementing or speeding up your code (though these may or may not be useful depending on your implementation strategy):

1. Suppose we have a matrix A and we want to extract a single element from each row, where the column of the element to be extracted from row i is stored in $y(i)$, where y is a row vector. We can use the `sub2ind()` function like this:

```
I=sub2ind(size(A), 1:size(A,1), y);
values = A(I);
```

This code will take each pair of indices (i, j) where i comes from the second argument and j comes from the corresponding element of the third argument, and compute the corresponding 1D index into A for the (i, j)'th element. So, $I(1)$ will be the index for the element at location $(1, y(1))$, and $I(2)$ will be the index for the element at $(2, y(2))$.

2. When you compute the predicted label probabilities $\hat{y}_k^{(i)} = \exp(\theta_{:,k}^\top x^{(i)}) / (\sum_{j=1}^K \exp(\theta_{:,j}^\top x^{(i)}))$, try to use matrix multiplications and `bsxfun` to speed up the computation. For example, once θ is in matrix form, you can compute the products for every example and the first 9 classes using $a = \theta^\top X$. (Recall that the 10th class is left out of θ, so that $a(10, :)$ is just assumed to be 0.)