Week 4: Logistic Regression

Instructor: Sergey Levine

1 Gradient Ascent on the Log-Likelihood

When we left off last time, we saw that we could represent a linear classifier in terms of a linear function $h(\mathbf{x}^i) \cdot \mathbf{w}$, where the probability of a binary label is given by

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{\exp(h(\mathbf{x}^i) \cdot \mathbf{w})}{\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1}$$

and

$$p(y = 0|\mathbf{x}, \mathbf{w}) = \frac{1}{\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1}$$

Let's write down the log-likelihood:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{N} \log p(y^i|\mathbf{x}^i, \mathbf{w}) \\
&= \sum_{i=1}^{N} \begin{cases} \text{if } y^i = 0\text{: } \log \frac{1}{\exp(h(\mathbf{x}^i)\cdot\mathbf{w})+1} \\ \text{if } y^i = 1\text{: } \log \frac{\exp(h(\mathbf{x}^i)\cdot\mathbf{w})}{\exp(h(\mathbf{x}^i)\cdot\mathbf{w})+1} \end{cases} \\
&= \sum_{i=1}^{N} \begin{cases} \text{if } y^i = 0\text{: } \log 1 - \log[\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1] \\ \text{if } y^i = 1\text{: } \log \exp(h(\mathbf{x}^i) \cdot \mathbf{w}) - \log[\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1] \end{cases} \\
&= \sum_{i=1}^{N} - \log[\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1] + \underbrace{\begin{cases} \text{if } y^i = 0\text{: } 0 \\ \text{if } y^i = 1\text{: } h(\mathbf{x}^i) \cdot \mathbf{w} \end{cases}}_{y^i h(\mathbf{x}^i)\cdot\mathbf{w}} \\
&= \sum_{i=1}^{N} \left(y^i h(\mathbf{x}^i) \cdot \mathbf{w} - \log[\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1] \right).
\end{aligned}$$

Now compute the gradient:

$$\begin{aligned}
\frac{d\mathcal{L}}{d\mathbf{w}} &= \sum_{i=1}^{N} \left(y^i h(\mathbf{x}^i) - \frac{h(\mathbf{x}^i) \exp(h(\mathbf{x}^i) \cdot \mathbf{w})}{\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1} \right) \\
&= \sum_{i=1}^{N} \left(h(\mathbf{x}^i) \left[y^i - \frac{\exp(h(\mathbf{x}^i) \cdot \mathbf{w})}{\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1} \right] \right).
\end{aligned}$$

Unfortunately, we can't simply set the gradient equal to zero and solve for $\mathbf{w}$ in closed form: there is actually no analytic formula for $\mathbf{w}$ in this case. Instead, we can use a procedure called gradient ascent (or gradient descent, if we instead minimize the negative log-likelihood).

Although we can't find an optimal setting of the parameter vector $\mathbf{w}$ in closed form, we know that, given some initial value $\mathbf{w}_0$, the log likelihood $\mathcal{L}(\mathbf{w})$ is going to increase if we modify $\mathbf{w}_0$ and move it in the direction given by $\frac{\mathcal{L}}{d\mathbf{w}}(\mathbf{w}_0)$, which we refer to as the gradient and sometimes write as $\nabla\mathcal{L}(\mathbf{w}_0)$. That is, we know that

$$\mathcal{L}(\mathbf{w}_0 + \alpha\nabla\mathcal{L}(\mathbf{w}_0)) \geq \mathcal{L}(\mathbf{w}_0).$$

for some small quantity α. We will use this intuition to devise an algorithm called gradient ascent. This algorithm climbs the gradient to maximize $\mathcal{L}(\mathbf{w})$, by repeatedly modifying $\mathbf{w}$ according to $\nabla\mathcal{L}(\mathbf{w})$ and recomputing the gradient. At the j^{th} iteration of gradient ascent, we compute the $(j+1)^{\text{th}}$ value of $\mathbf{w}$ according to:

$$\mathbf{w}^{(j+1)} \leftarrow \mathbf{w}^{(j)} + \alpha\nabla\mathcal{L}(\mathbf{w}^{(j)}).$$

The size of the step that we take α is referred to as a learning rate or a step size, and we typically use a small positive constant, such as 10^{-2}. Choosing this constant takes some care, as an excessively high α can cause us to overshoot the optimal $\mathbf{w}$, and a low α can cause learning to take a very long time. We can try a few different values and choose the one that produces the best solution (we don't need a validation set for this, we can just use the likelihood of the training set directly).

To derive the full gradient ascent update, let's first revisit the equation for the gradient:

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{i=1}^{N}\left(h(\mathbf{x}^i)\left[y^i - \frac{\exp(h(\mathbf{x}^i)\cdot\mathbf{w})}{\exp(h(\mathbf{x}^i)\cdot\mathbf{w})+1}\right]\right).$$

Note that this is precisely equal to

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{i=1}^{N} h(\mathbf{x}^i)\left[y^i - p(y=1|\mathbf{x}^i,\mathbf{w})\right].$$

So we can write the gradient ascent update for logistic regression as

$$\mathbf{w}^{(j+1)} \leftarrow \mathbf{w}^{(j)} + \alpha\sum_{i=1}^{N} h(\mathbf{x}^i)\left[y^i - p(y=1|\mathbf{x}^i,\mathbf{w}^{(j)})\right].$$

This has a natural geometric intuition: for each datapoint in the dataset, we move $\mathbf{w}$ toward its attributes $h(\mathbf{x}^i)$ if the label is 1 but $p(y=1|\mathbf{x}^i,\mathbf{w})$ is too low, or away from $h(\mathbf{x}^i)$ if the label is 0 but $p(y=1|\mathbf{x}^i,\mathbf{w})$ is too high.

The log likelihood for logistic regression is convex: that means that there is only one unique optimum, and we can reach that optimum by following the gradient. That means that gradient ascent converges to the optimal solution, so

long as we choose a reasonable learning rate. Theoretically, so long as the learning rate decreases at a particular rate, we are guaranteed to eventually reach the global optimum. In practice, we might reduce the learning rate gradually during learning, for example by multiplying it by 0.9 ever 100 or 1000 iterations.

We can also use more advanced optimization algorithms, such as conjugate gradient or LBFGS. These algorithms approximate Newton's method by estimating the curvature of the log likelihood function, and typically employ a line search to set the learning rate automatically.

2 MAP Estimation

Like any other learning method, logistic regression is liable to overfit when the training set is too small or the hypothesis class is too complex (for example when too many different features are included). When logistic regression overfits, it tends to produce very large weights. Note that the decision boundary $f(\mathbf{x}) = h(\mathbf{x}) \cdot \mathbf{w} = 0$ remains in the same place as we scale $\mathbf{w}$ by some constant to get $\beta\mathbf{w}$, for any $\beta \in \mathbb{R}$. However, as $\mathbf{w}$ increase in magnitude, the likelihood of correctly classified training examples near the boundary will increase, since their probabilities will be closer to 1.0. Of course, this can produce overfitting if the number of such points is too small to localize the boundary correctly: it's better in this case to keep the boundary fuzzy and soft to accurately reflect our uncertainty, instead of fitting every tiny detail of the training points with a sharp and complex decision boundary.

Question. How can we mitigate overfitting?

Answer. Just like with linear regression and naïve Bayes, we can use maximum a posteriori (MAP) estimation to put a prior on the parameters $\mathbf{w}$. We can use the same priors as with linear regression: a Gaussian prior or a Laplace (absolute value) prior. Let's discuss the Gaussian prior:

$$p(\mathbf{w}) = \prod_{k=1}^{K} \frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{\mathbf{w}_k^2}{2\sigma_0^2}}$$

Writing the equation for the log posterior, we get

$$\log p(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \log p(y^i|\mathbf{x}^i, \mathbf{w}) - \frac{1}{2\sigma_0^2} \sum_{k=1}^{K} \mathbf{w}_k^2 + \text{const.}$$

Like we did with linear regression, we'll typically set a regularization constant λ instead of using σ_0. Making this change and substituting in the equation for the log-likelihood, we get

$$\mathcal{L}(\mathbf{w}) = \log p(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \left(y^i h(\mathbf{x}^i) \cdot \mathbf{w} - \log[\exp(h(\mathbf{x}^i) \cdot \mathbf{w}) + 1] \right) + \frac{\lambda}{2} \sum_{k=1}^{K} \mathbf{w}_k^2,$$

and the gradient is given by

$$\nabla \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} h(\mathbf{x}^i)[y^i - p(y = 1|\mathbf{x}^i, \mathbf{w})] - \lambda \mathbf{w}$$

So we see that with a very simple modification to the gradient, we can regularize the weights and prevent them from becoming very large.

Question. What is the gradient ascent update rule for MAP?

Answer. Just like before, we have

$$\mathbf{w}^{(j+1)} \leftarrow \mathbf{w}^{(j)} + \alpha \nabla \mathcal{L}(\mathbf{w}^{(j)})$$

and therefore

$$\mathbf{w}^{(j+1)} \leftarrow \mathbf{w}^{(j)} + \alpha \left(\sum_{i=1}^{N} h(\mathbf{x}^i) \left[y^i - p(y = 1|\mathbf{x}^i, \mathbf{w}^{(j)}) \right] - \lambda \mathbf{w}^{(j)} \right).$$

3 Multiclass Classification

What if we have more than two possible values of y? That is, $y \in \{1, \ldots, L_y\}$. We can still use logistic regression, but now instead of a weight vector $\mathbf{w}$, we have a weight matrix $\mathbf{W}$, where each column of this matrix is a different weight vector:

$$\mathbf{W} = \left[\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{L_y} \right].$$

Then, we simply set the probability of each class according to:

$$p(y = j|\mathbf{x}, \mathbf{W}) \propto \exp(h(\mathbf{x}) \cdot \mathbf{w}_j).$$

Since we know the probabilities must sum to 1, we know that

$$p(y = j|\mathbf{x}, \mathbf{W}) = \frac{\exp(h(\mathbf{x}) \cdot \mathbf{w}_j)}{\sum_{j'=1}^{L_y} \exp(h(\mathbf{x}) \cdot \mathbf{w}_{j'})}.$$

We can then compute the gradient of the log-likelihood in exactly the same way as before and perform gradient ascent on the entire matrix $\mathbf{W}$.

Note that this representation is overcomplete. Indeed, we can represent an equivalent multiclass classifier using a matrix $\mathbf{W}$ with only $L_y - 1$ columns, since we know that the probability of the last label $y = L_y$ is given by

$$p(y = L_y|\mathbf{x}, \mathbf{W}) = 1 - \sum_{j=1}^{L_y - 1} p(y = j|\mathbf{x}, \mathbf{W})$$

Deriving the corresponding equations for $p(y = j|\mathbf{x}, \mathbf{W})$ and the gradient is left as an exercise.