Lecture 5: More on logistic regression. Second-order methods. Kernels

- Logistic regression
- Regularization
- Kernelizing linear methods

Recall: Logistic regression

- Hypothesis is a logistic function of a linear combination of inputs:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$

- We interpret $h(\mathbf{x})$ as $P(y = 1|\mathbf{x})$
- Then the log-odds ratio, $\ln\left(\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\right) = \mathbf{w}^T\mathbf{x}$ is linear
- Optimizes the *cross-entropy error function* :

$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i))\right)$$

using gradient descent (or ascent on the log likelihood of the data)

Maximization procedure: Gradient ascent

- First we compute the gradient of $\log L(\mathbf{w})$ wrt $\mathbf{w}$
- The update rule is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla \log L(\mathbf{w}) = \mathbf{w} + \alpha \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x_i}))\mathbf{x_i} = \mathbf{w} + \alpha \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}})$$

  where $\alpha \in (0, 1)$ is a step-size or learning rate parameter
- If one uses features of the input, we have:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{\Phi}^T(\mathbf{y} - \hat{\mathbf{y}})$$

- The step size α is a parameter for which we have to choose a "good" value

Roadmap

- If the cost function is convex then gradient descent will converge to the optimal solution for an appropriate choice of the learning rates.
- We will show that the cross-entropy error function is convex.
- We will see how we can use a second-order method to choose "optimal" learning rates.

Convexity

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, $\lambda \in [0, 1]$:

$$f(\lambda \mathbf{a} + (1 - \lambda)\mathbf{b}) \leq \lambda f(\mathbf{a}) + (1 - \lambda) f(\mathbf{b}).$$

- If f and g are convex functions, $\alpha f + \beta g$ is also convex for any real numbers α and β.

Characterizations of convexity

- *First-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{for all } \mathbf{a}, \mathbf{b}: \quad f(\mathbf{a}) \geq f(\mathbf{b}) + \nabla f(\mathbf{b})^\top (\mathbf{a} - \mathbf{b})$$

  (the function is globally above the tangent at $\mathbf{b}$).
- *Second-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{the Hessian of } f \text{ is positive semi-definite.}$$

  The Hessian contains the second-order derivatives of f:

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

  It is positive semi-definite if $\mathbf{a}^\top \mathbf{H} \mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{R}^d$.

Convexity of the cost function

$$J(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))\right)$$

where $\sigma(z) = 1/(1 + e^{-z})$ (check that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$).

- We show that $-\log \sigma(\mathbf{w}^\top \mathbf{x})$ and $-\log(1 - \sigma(\mathbf{w}^\top \mathbf{x}))$ are convex in $\mathbf{w}$:

$$\nabla_{\mathbf{w}} \left(-\log \sigma(\mathbf{w}^\top \mathbf{x})\right) = -\frac{\nabla_{\mathbf{w}} \left(\sigma(\mathbf{w}^\top \mathbf{x})\right)}{\sigma(\mathbf{w}^\top \mathbf{x})} = -\frac{\sigma'(\mathbf{w}^\top \mathbf{x}) \nabla_{\mathbf{w}} (\mathbf{w}^\top \mathbf{x})}{\sigma(\mathbf{w}^\top \mathbf{x})} = (\sigma(\mathbf{w}^\top \mathbf{x}) - 1)\mathbf{x}$$

$$\nabla^2_{\mathbf{w}} \left(-\log \sigma(\mathbf{w}^\top \mathbf{x})\right) = \nabla_{\mathbf{w}} \left(\sigma(\mathbf{w}^\top \mathbf{x})\mathbf{x}\right) = \sigma(\mathbf{w}^\top \mathbf{x})(1 - \sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}\mathbf{x}^\top$$

$\Rightarrow$ It is easy to check that this matrix is positive semi-definite for any $\mathbf{x}$.

Convexity of the cost function

$$J(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i))\right)$$

where $\sigma(z) = 1/(1+e^{-z})$ (check that $\sigma'(z) = \sigma(z)(1-\sigma(z))$).

- Similarly you can show that

$$\nabla_{\mathbf{w}} \left(-\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))\right) = \sigma(\mathbf{w}^\top \mathbf{x})\mathbf{x}$$

$$\nabla^2_{\mathbf{w}} \left(-\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))\right) = \sigma(\mathbf{w}^\top \mathbf{x})(1-\sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}\mathbf{x}^\top$$

$\Rightarrow$ $J(\mathbf{w})$ is convex in $\mathbf{w}$.

$\Rightarrow$ The gradient of J is $\mathbf{X}^\top(\hat{\mathbf{y}} - \mathbf{y})$ where $\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i) = h(\mathbf{x}_i)$.

$\Rightarrow$ The Hessian of J is $\mathbf{X}^\top \mathbf{R} \mathbf{X}$ where $\mathbf{R}$ is diagonal with entries $\mathbf{R}_{i,i} = h(\mathbf{x}_i)(1-h(\mathbf{x}_i))$.

Another algorithm for optimization

- Recall Newton's method for finding the zero of a function $g : \mathbb{R} \to \mathbb{R}$
- At point w^i, approximate the function by a straight line (its tangent)
- Solve the linear equation for where the tangent equals 0, and move the parameter to this point:

$$w^{i+1} = w^i - \frac{g(w^i)}{g'(w^i)}$$

Application to machine learning

- Suppose for simplicity that the error function J has only one parameter
- We want to optimize J, so we can apply Newton's method to find the zeros of $J' = \frac{d}{dw}J$
- We obtain the iteration:

$$w^{i+1} = w^i - \frac{J'(w^i)}{J''(w^i)}$$

- Note that there is *no step size parameter*!
- This is a *second-order method*, because it requires computing the second derivative
- But, if our error function is quadratic, this will find the global optimum in one step!

Second-order methods: Multivariate setting

- If we have an error function J that depends on many variables, we can compute the *Hessian matrix*, which contains the second-order derivatives of J:

$$\mathbf{H}_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

- The inverse of the Hessian gives the "optimal" learning rates
- The weights are updated as:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1} \nabla_{\mathbf{w}} J$$

- This is also called Newton-Raphson method for logistic regression, or Fisher scoring

Which method is better?

- Newton's method usually requires significantly fewer iterations than gradient descent
- Computing the Hessian requires a batch of data, so there is no natural on-line algorithm
- Inverting the Hessian explicitly is expensive, but almost never necessary
- Computing the product of a Hessian with a vector can be done in linear time (Pearlmutter, 1993), which helps also to compute the product of the inverse Hessian with a vector without explicitly computing $\mathbf{H}$

Newton-Raphson for logistic regression

- Leads to a nice algorithm called *iteratively reweighted least squares* (or iterative recursive least squares)
- The Hessian has the form:

$$\mathbf{H} = \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi}$$

  where $\mathbf{R}$ is the diagonal matrix of $h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))$ (you can check that this is the form of the second derivative).
- The weight update becomes:

$$\mathbf{w} \leftarrow \mathbf{w} - (\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top (\hat{\mathbf{y}} - \mathbf{y})$$

  which can be rewritten as the solution of a weighted least square problem:

$$\mathbf{w} \leftarrow (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{R} (\mathbf{\Phi}\mathbf{w} - \mathbf{R}^{-1} (\hat{\mathbf{y}} - \mathbf{y}))$$

Regularization for logistic regression

- One can do regularization for logistic regression just like in the case of linear regression
- Recall regularization makes a statement about the weights, so does not affect the error function
- Eg: L_2 regularization will have the optimization criterions:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

Probabilistic view of logistic regression

- Consider the additive noise model we discussed before:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon$$

  where ϵ are drawn iid from some distribution
- At first glance, log reg does not fit very well
- We will instead think of a latent variable $\hat{y}_i$ such that:

$$\hat{y}_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon$$

- Then the output is generated as:

$$y_i = 1 \text{ iff } \hat{y}_i > 0$$

Generalized Linear Models

- Logistic regression is a special case of a generalized linear model:

$$E[Y \mid \mathbf{x}] = g^{-1}(\mathbf{w}^\top \mathbf{x}).$$

  g is called the *link function*, it relates the mean of the response to the linear predictor.
- Linear regression: $E[Y \mid \mathbf{x}] = E[\mathbf{w}^\top \mathbf{x} + \varepsilon \mid \mathbf{x}] = \mathbf{w}^\top \mathbf{x}$ (g is the identity).
- Logistic regression: $E[Y \mid \mathbf{x}] = P(Y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$
- Poisson regression: $E[Y \mid \mathbf{x}] = \exp(\mathbf{w}^\top \mathbf{x})$ (for count data).
- ...

Linear regression with feature vectors revisited

- Find the weight vector $\mathbf{w}$ which minimizes the (regularized) error function:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Suppose instead of the closed-form solution, we just take the gradient and rearrange the terms
- The solution takes the form:

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{i=1}^{m}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)\phi(\mathbf{x}_i) = \sum_{i=1}^{m} a_i\phi(\mathbf{x}_i) = \mathbf{\Phi}^T\mathbf{a}$$

  where $\mathbf{a}$ is a vector of size m (number of instances) with $a_i = -\frac{1}{\lambda}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)$
- Main idea: *use $\mathbf{a}$ instead of $\mathbf{w}$ as parameter vector*

Re-writing the error function

- Instead of $J(\mathbf{w})$ we have $J(\mathbf{a})$:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a} - \mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a}$$

- Denote $\mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K}$
- Hence, we can re-write this as:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}$$

- This is quadratic in $\mathbf{a}$, and we can set the gradient to 0 and solve.

Dual-view regression

- By setting the gradient to 0 we get:

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

- Note that this is similar to re-formulating a weight vector in terms of a linear combination of instances
- Again, the *feature mapping is not needed* either to learn or to make predictions!
- This approach is useful if the feature space is very large

Kernel functions

- Whenever a learning algorithm can be written in terms of dot-products, it can be generalized to kernels.
- A *kernel* is any function $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping ϕ:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) \text{ for some } \phi.$$

- Conversely, by choosing feature mapping ϕ, we implicitly choose a kernel function
- Recall that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos \angle(\mathbf{x}_1, \mathbf{x}_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of *similarity*.

Example: Quadratic kernel

- Let $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.
- Is this a kernel?

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) = \sum_{i,j \in \{1 \ldots n\}} x_i z_i x_j z_j \\ &= \sum_{i,j \in \{1 \ldots n\}} (x_i x_j)(z_i z_j) \end{aligned}$$

- Hence, it is a kernel, with feature mapping:

$$\phi(\mathbf{x}) = \langle x_1^2,\ x_1 x_2,\ \ldots,\ x_1 x_n,\ x_2 x_1,\ x_2^2,\ \ldots,\ x_n^2 \rangle$$

  Feature vector includes all squares of elements and all cross terms.
- Note that computing ϕ takes $O(n^2)$ but *computing K takes only $O(n)$*!