Optimization

Optimization

- We have already discussed the fact that there is an analytic solution for the least squares loss in a linear regression
- We derive the analytic solution in the obvious way by taking the derivative
- This specific case is of little interest here and we put it aside; rather, we are interested in how we might find a solution for logistic regression
- Here we have no choice but to appeal to a search, or, more precisely, an **optimization**
- The difference between the two terms is subtle but important here, as an optimization problem refers to one in which our algorithm attempts to find the "best-scoring" solution that it can, whereas the more general term search includes cases where a solution is either right or wrong (e.g., Sudoku)
- As such, it is worthwhile to point out that, in general, we have no way of assuring that the best solution we are able to *find* is the best solution there *is*
- I should say, as an aside, that, in fact, for logistic regression, we *can* prove that this is the case; this is because logistic regression is what is called a *convex* problem; this means, intuitively, that there is a "bowl" that contains the solutions
- Not all convex problems have analytic solutions, but there are effective methods for finding a global optimum for convex problems
- Scikit-learn and other packages decide not to make use of these methods (at least not by default) and instead appeal to more general optimization methods

Loss function

- Instead of a "score" function we usually talk about a **utility** or **loss** function - these terms encode the direction of what is desirable: a utility function is better for higher values, a loss function is better for lower values
- Usually optimization is formulated in terms of loss functions, that is, functions to be minimized

- We have already seen an example of a loss function, namely the sum of squared error function used for linear regression
- For the logistic regression we maximize the likelihood, which is the following straightforward formula:

$$\prod_{i:y_i=1} p(y_i = 1|x_i) \prod_{i':y_{i'}=0} (1 - p(y_{i'}|x_{i'}))$$

- This is equivalent to minimizing the negative log likelihood, as follows:

$$-\sum_{i:y_i=1} \log p(y_i = 1|x_i) - \sum_{i':y_{i'}=0} \log(1 - p(y_{i'} = 1|x_{i'}))$$

- Implementing this naively would have you splitting off the different cases using a conditional, but you can do it simpler assuming a 1-0 encoding of the response, as follows (where I now write just p_i instead of the complicated expression):

$$-\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

- Recalling that what we are trying to do is build a model that estimates $p()$ via some values $logistic(z_i)$ where $z_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik}$, which are a function of the β parameters - the βs are what we are searching for
- Thus, substituting in $logistic(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik})$ into the NLL above (which I will let you do on paper or in your head so as not to make a mess in the PDF), we have a function of the parameters that tells us how bad they are
- The data is a constant
- Returning briefly to the actual form of the function itself, you might notice that it is in theory possible to stumble upon $\log 0$
- Although you might think this is *im*-possible given that the logistic only approaches, and never actually reaches, zero or one, it does happen
- For this to happen, there needs to be a region (as a function of x) where the data is literally just all 1s or all 0s, in which case it can happen that the model pushes the z's to be "numerically infinite", that is, so large in magnitude that, up to the precision of the machine, p actually is 1 or 0
- This *may* happen in the case described above, and it *will* happen if the problem is **perfectly separable**, meaning that there is **no overlap at all** between the two classes
- Forget the perfectly separable case for today, though it can be addressed with regularization
- $\log 0$ happens often enough in logistic regression that we usually switch to another form of the loss function, as follows:

$$\sum_i \log(1 + \exp z_i) - y_i z_i$$

- Why? Start by rearranging the last version of the expression, your goal being to get two terms, one with a y and one without a y
- Henceforth I am going to refer to the parameters we are searching for as a vector θ and I am going to call the loss function f
- Often we call the loss function ℓ, but let's not get fancy

Newton-type methods

- The intuition behind Newton-type methods for optimization, including the **L**ow-memory **B**royden-**F**letcher-**G**oldfarb-**S**hanno (L-BFGS) algorithm that scikit-learn uses by default for logistic regression, is that, if the function were not just convex but **quadratic**, it would have an obvious analytic minimum
- So what we do is to find a quadratic approximation to f such that, at the current value of θ, we have (a) the same value of f (b) the same gradient (set of partial derivatives) and (c) the same curvature as represented by the matrix of second-order partial derivatives
- Consider two things: first, how do we find these derivatives (what approach might we take)?
- Second, how complex are the derivatives themselves? Consider just the number of values that we have to calculate as a first approximation of computational complexity
- The answer to the second question is easy: if d is the dimension of θ, the gradient has d values and the matrix of partial derivatives has d^2 values
- This means we are
- There is a dumb answer to the first question: numerical differentiation
- Numerical differentiation, it turns out, is not very good for large d as (1) large d means many roundoff errors and (2) large d means lots and lots and lots and lots of finite differences
- Furthermore, calculating the Hessian matrix is just intrinsically complex
- However, that is how Newton-type methods work, so let's embrace it

Exercise

Instead of using scikit-learn, fit the logistic regression for the *Default* data set yourself using L-BFGS, using the scipy package, using scipy.optimize.minimize (see documentation at https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html). No need to implement L-BFGS, just take it off the shelf.