4 ELR Learning Algorithm

Given the intractability of computing the optimal CPtable entries in general, we defined a simple gradient-ascent algorithm, ELR, that attempts to improve the empirical score $\widehat{\mathrm{LCL}}(\Theta)$ by changing the values of each CPtable entry $\theta_{d|\mathbf{f}}$. (Of course, this will only find, at best, a *local* optimum.) To incorporate the constraints $\theta_{d|\mathbf{f}} \geq 0$ and $\sum_d \theta_{d|\mathbf{f}} = 1$, we used the different set of parameters, "$\beta_{d|\mathbf{f}}$", where each

$$\theta_{d|\mathbf{f}} \quad = \quad \frac{e^{\beta_{d|\mathbf{f}}}}{\sum_{d'} e^{\beta_{d'|\mathbf{f}}}} \; . \tag{1}$$

As the β_is sweep over the reals, the corresponding $\theta_{d_i|\mathbf{f}}$'s will satisfy the appropriate constraints. (In the naïve-bayes case, this corresponds to what many logistic regression algorithms would do, albeit with different parameters [Jor95]: Find α, χ that optimize $P_{\alpha,\chi}(C = c \,|\, \mathbf{E} = \mathbf{e}) \;=\; e^{\alpha_c + \chi_c \cdot \mathbf{e}} / \sum_j e^{\alpha_j + \chi_j \cdot \mathbf{e}}$.[1] Recall that our goal is a more general algorithm — one that can deal with *arbitrary* structures; see Equation 2.)

Like all such algorithms, ELR is basically

$$\begin{array}{l} \text{Initialize } \beta^{(0)} \\ \text{For } k = 1..m \\ \quad \beta^{(k+1)} \quad := \quad \beta^{(k)} \; + \; \alpha^{(k)} \times \mathbf{d}^{(k)} \end{array} \tag{2}$$

where $\beta^{(k)}$ represents the set of parameters at iteration k. In a simple gradient ascent approach, we would set $\mathbf{d}^{(k)}$ to be the total derivative with respect to the given set of labeled queries, $\nabla \widehat{LCL} \;=\; \langle \frac{\partial \, \widehat{\mathrm{LCL}}^{(S)}(\Theta)}{\partial \, \beta_{d|\mathbf{f}}} \rangle_{d,\mathbf{f}}$, which is the sum of the individual derivatives from each labeled training case $\langle \mathbf{e}; c \rangle$:

$$\frac{\partial \, \widehat{\mathrm{LCL}}^{(S)}(\Theta)}{\partial \, \beta_{d|\mathbf{f}}} \quad = \quad \sum_{\langle \mathbf{e}, c \rangle \in S} \frac{\partial \, \widehat{\mathrm{LCL}}^{(\langle \mathbf{e}, c \rangle)}(\Theta)}{\partial \, \beta_{d|\mathbf{f}}} \; .$$

This requires...

Proposition 1 *For the labeled training case* $\langle \mathbf{e}, c \rangle$ *and each "softmax" parameter* $\beta_{d|\mathbf{f}}$,

$$\frac{\partial \, \widehat{LCL}^{(\langle \mathbf{e}, c \rangle)}(\Theta)}{\partial \, \beta_{d|\mathbf{f}}} \;=\; [P_\Theta(d, \mathbf{f} \,|\, \mathbf{e}, \, c) - P_\Theta(d, \mathbf{f} \,|\, \mathbf{e})] \;-\; \theta_{d|\mathbf{f}} \, [P_\Theta(\mathbf{f} \,|\, c, \, \mathbf{e}) - P_\Theta(\mathbf{f} \,|\, \mathbf{e})] \; .$$

Recall $P_\Theta(x \,|\, y)$ refers to conditional probability of x given a particular value of y, for the parameter setting corresponding to $\Theta \sim \langle \beta_{d|\mathbf{f}} \rangle$. Notice this expression is well-defined for any set of evidence variables $\mathbf{E}$ — which can be all "non-C" variables (corresponding to a complete data tuple), or any subset, including the empty $\mathbf{E} = \{\}$.

To work effectively, ELR incorporates four instantiations/modifications to the basic gradient ascent idea, dealing with

1. the initial values $\beta^{(0)}$ ("plug-in parameters"),
2. the direction of the modification $\mathbf{d}^{(k)}$ (conjugate gradient),
3. the magnitude of the change $\alpha^{(k)}$ (line search) and
4. the stopping criteria m ("cross tuning").

Minka [Min01] has shown that the middle two ideas, conjugate gradient and line-search [PFTV02], are effective for the standard logistic regression task.

Begin with Plug-In Parameters: We must first initialize the parameters, $\beta^{(0)}$. One common approach is to set $\beta^{(0)}$ to small, randomly selected, values; another is to begin with the values specified by the generative approach — *i.e.*, using frequency estimates (OFE; Equation 12) in the complete data case, and a simple variant otherwise.[2]

[1] While the obvious tabular representation of the CPtables involves more parameters than appear in this logistic regression model, these extra BN-parameters are redundant.

[2] To compute the value for the $\beta_{d|\mathbf{f}}$ parameter, use frequency estimates but only over the training instances that specify the values of all $\{D\} \cup \mathbf{F}$ variables.

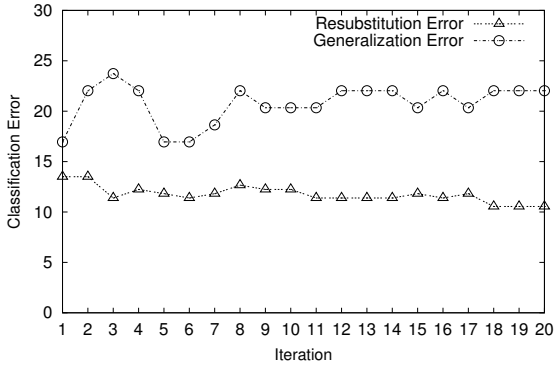

Figure 1: Comparing Training-Set Error with Test-Set Error, as function of Iteration

Our empirical evidence shows that this second approach works better, especially for small samples. These easy-to-compute generative starting values are often used to initialize parameters for discriminative tasks, and called "plug-in parameters" [Rip96].

Conjugate Gradient: Standard gradient ascent algorithms may require a great many iterations, especially for functions that have long, narrow valley structures. The *conjugate gradient method* addresses this problem by ascending along *conjugate directions*, rather than simply the local gradient. As this means that the directions that have already been optimized, stay optimized, this method often requires far fewer steps uphill to reach the local optimum [HDB96].

In particular, ELR uses the *Polak-Rebiere* formula to update its search direction. The initial search direction is given by:

$$\mathbf{d}^{(0)} \quad = \quad \nabla \widehat{LCL}_{(0)}$$

On subsequent iterations,

$$\mathbf{d}^{(k)} \quad = \quad \nabla \widehat{LCL}_{(k)} \; - \; \frac{(\nabla \widehat{LCL}_{(k)} - \nabla \widehat{LCL}_{(k-1)}) \; \cdot \; \nabla \widehat{LCL}_{(k)}}{\nabla \widehat{LCL}_{(k-1)} \; \cdot \; \nabla \widehat{LCL}_{(k-1)}} \, \mathbf{d}^{(k-1)}$$

where the "$\cdot$" is the vector dot-product. Hence, the current update direction is formed by "subtracting off" the previous direction from the current derivative.

Line Search: ELR will ascend in the $\mathbf{d}^{(k)}$ direction; the next challenge is deciding how far to move — *i.e.*, in computing the $\alpha^{(k)} \in \Re^+$ from Equation 2.

The good news is, as $\beta = \beta^{(k)}$ and $\mathbf{d} = \mathbf{d}^{(k)}$ are fixed, this is a one-dimensional search: first α^* that maximizes $f(\alpha) = \widehat{\mathrm{LCL}}^{(S)}(\beta + \alpha_i \times \mathbf{d})$. ELR therefore uses a standard technique, *Brent*'s iterative line search procedure (a hybrid combination of the linear Golden Section search [HDB96] and a quadratic interpolation), which has proven to be very effective at finding the optimal value for $\alpha^{(k)}$ [Bis98]. In essense, this method first finds three α_i values, and computes the associated function values $\langle \langle \alpha_i, \; f(\alpha_i) \rangle \rangle_{i=1,2,3}$. It then assumes the $f(\cdot)$ function is (locally) quadratic, and so fits this trio of points to a second degree polynomial. It then finds the α^* value that minimizes this quadratic, replaces one of the three original α_i values with this $\langle \alpha^*, \; f(\alpha^*) \rangle$ pair, and iterates using the new trio of points. See [Bis98] for details.

Cross tuning (stopping time): The final issue is deciding when to stop; *i.e.*, determining the value of m for Equation 2. A naïve algorithm would just compute the training-set error (Equation 9) at each iteration, and stop when that error measure appears at a local optimum — *i.e.*, when the error appears to be going up. The graph in Figure 1 shows both training-set and test-set error on a particular dataset (here "CLEVE"; see next section), at each iteration. If we used this simple approach, we would stop on the third iteration; the generalization error shows that these parameters are very bad — in fact, they seem almost the worst values!

To avoid this, we use a variant of cross validation, which we call "cross tuning", to estimate the optimal number of iterations. Here, we divide the *training* data into $\ell = 5$ partitions, then for each fold, run the gradient ascent for remaining $4/5$ of the data, but evaluating the quality of the result (on each iteration) on the fold. (This produces a graph

like the "Generalization Error" line in Figure 1.) We then determine, for each fold, when we should have stopped — here, it would be on $k = 5$, as that is the global optimum for this fold. We then set m to be the median values over these folds. When using all of the data to produce the final $\beta_{d|\mathbf{f}}$ values, we iterate exactly m times. The website [Gre04, #CrossTuning] presents empirical evidence that cross-tuning is important, especially for complex models.

References

[Bis98] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford, 1998.

[Gre04] 2004. http://www.cs.ualberta.ca/∼greiner/ELR.

[HDB96] M. Hagan, H. Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, Boston, MA, 1996.

[Jor95] M. Jordan. Why the logistic function? a tutorial discussion on probabilities and neural networks, 1995.

[Min01] Tom Minka. Algorithms for maximum-likelihood logistic regression. Technical report, CMU CALD, 2001. http://www.stat.cmu.edu/∼minka/papers/logreg/minka-logreg.pdf.

[PFTV02] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge, 2002. http://www.nr.com/.

[Rip96] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.