Cost Learning Approaches

Bill McDowell
forkunited@gmail.com

Spring 2014

This document contains approaches to cost learning that we considered prior to the approach described in the unpublished "Learning Not to Try Too Hard" paper. We tried the approaches described in the current document in roughly chronological order. The last approach is an alternative description of the approach given in the unpublished paper.

1 Cost Learning SVM

We have implemented a cost learning SVM that uses Adagrad to minimize the following objective function:[1]

$$\begin{aligned} &\min_{\mathbf{v}\geq 0,\mathbf{w},\mathbf{b},\|\mathbf{v}\|_1=1} \lambda_2\|\mathbf{w}\|_2^2+\lambda_1\|\mathbf{w}\|_1 \\ &+\sum_{i=1}^{N}\Bigg(-\mathbf{w}^\top\mathbf{g}(\mathbf{x}_i,\mathbf{y}_i)-\mathbf{b}^\top\mathbf{l}(\mathbf{y}_i)+\max_{\mathbf{y}\in\mathcal{Y}_{x_i}}\Big(\mathbf{w}^\top\mathbf{g}(\mathbf{x}_i,\mathbf{y})+\mathbf{b}^\top\mathbf{l}(\mathbf{y})+\mathbf{v}^\top\mathbf{c}(\mathbf{x}_i,\mathbf{y}_i,\mathbf{y})\Big)\Bigg) \end{aligned} \tag{1}$$

And makes a prediction $\mathbf{y}$ for input $\mathbf{x}$ using the learned weights $\mathbf{w}$ according to:

$$\mathbf{y}=\operatorname*{argmax}_{\mathbf{y}'\in\mathcal{Y}_x}\Big(\mathbf{w}^\top\mathbf{g}(\mathbf{x},\mathbf{y}')+\mathbf{b}^\top\mathbf{l}(\mathbf{y}')\Big) \tag{2}$$

Where the components of $\mathbf{w}$ are the regularized weights on features $\mathbf{g}$, the components of $\mathbf{b}$ are biases on label counts $\mathbf{l}$, and the components of $\mathbf{v}$ are cost weights on factored cost $\mathbf{c}$. The minimization of objective 1 results in predictive feature weights $\mathbf{w}$ and a cost function determined by cost weights $\mathbf{v}$. The learned cost function can be written more explicitly as:

$$\begin{aligned} c(\mathbf{x},\mathbf{y},\mathbf{y}') &= \mathbf{v}^\top\mathbf{c}(\mathbf{x},\mathbf{y},\mathbf{y}') \\ &= \sum_{S\in\mathcal{S}} v_S c_S(\mathbf{x},\mathbf{y},\mathbf{y}') \\ &= K\sum_{S\in\mathcal{S}} v_S\mathbf{1}((\mathbf{x},\mathbf{y},\mathbf{y}')\in S) \end{aligned} \tag{3}$$

Where $\mathcal{S}$ contains prediction classes $S\subseteq\mathcal{X}\times\mathcal{Y}^2$ of input, output, prediction triples according to which the cost function c factors, and K is a cost constant determining the magnitude of c. For a concrete example, if we assume $\mathcal{Y}$ is is a non-structured space of labels, we could choose $\mathcal{S}$ as:

[1] The objective requires projecting the cost weights to the simplex after each update with Adagrad. Our implementation uses a modification of the algorithm described at `http://machinelearning.org/archive/icml2008/papers/361.pdf` to perform this projection under the Mahalanobis norm required by Adagrad. We didn't have time to write up the derivation for this modification, but it shouldn't be too hard to figure out again if you need it in the future.

$$\mathcal{S}_{[\mathcal{Y}]^2} = \{S_{y,y'} | y, y' \in \mathcal{Y}\} \tag{4}$$

With:

$$S_{y,y'} = \{(\mathbf{x}, l, l') | (l = y \text{ and } l' = y') \text{ or } (l = y' \text{ and } l = y)\} \tag{5}$$

Factoring the cost by $\mathcal{S}_{[\mathcal{Y}]^2}$ allows the model to decide which conflations between pairs of labels should cost the most. For example, for $y_1, y_2, y_3 \in \mathcal{Y}$, the model might choose that conflating y_1 and y_2 is a costlier mistake than conflating y_2 with y_3, which would be reflected by $v_{S_{y_1,y_2}} > v_{S_{y_2,y_3}}$. Alternatively, we might choose $\mathcal{S}$ to so that the cost factors by sets of input, output, prediction triples that depend on features of the input, which would allow the model to assign costs to predictions for certain values of input features.

Intuitively, the goal of the cost function learning is to allow the model to choose the costs for classes of incorrect predictions based on how difficult it is to shrink them–with larger costs for easier classes and smaller costs for harder classes. This choice of costs gives the model the freedom to shift its weights to correctly classify more difficult examples within the easier prediction classes while giving up on examples in the harder prediction classes. Such a policy is particularly useful when there are classes of incorrect predictions that are difficult to resolve due to unreliably annotated output labels or a choice of model that is insufficient for the task.

The intuition for the cost function learning would be reflected by cost weights $\mathbf{v}^*$ in learned model M^* that have the property:

$$\forall S, S' \in \mathcal{S}, v^*_S \leq v^*_{S'} \iff D_S(M^*) \geq D'_S(M^*) \tag{6}$$

Where $D_S(M^*)$ is some measure of the difficulty of resolving errors from incorrect prediction class S for model M^*. We also need $\mathbf{v}^* \geq 0$ to ensure a non-negative cost-function, and we'd prefer that $\mathbf{v}^* \leq 1$ so that $\mathbf{v}^*$ is easy to interpret.

Notice that objective function 1 meets the condition that $0 \leq \mathbf{v}^1 \leq 1$. However, it will be minimized by a $\mathbf{v}^1$ for which the following tends to be true: [2]

$$\forall S, S' \in \mathcal{S}, v^1_S \leq {v^1}_{S'} \iff |S| \geq |S'| \tag{7}$$

If $|S|$ is good to use as difficulty measure D_S, then objective function 1 produces $\mathbf{v}^1$ that satisfies property 6. This seems somewhat plausible considering that if the model makes a large number of incorrect predictions of class S, then S is difficult for the model to shrink or resolve. Unfortunately, for many data sets and choices of $\mathcal{S}$, the size of each $S \in \mathcal{S}$ is inherently biased by the data regardless of its difficulty. For example, for $S_{y,y'} \in \mathcal{S}_{[\mathcal{Y}]^2}$, $|S_{y,y'}|$ is biased by the number of examples in the data which have output labels y and y'–if there are few training examples of labels $y, y' \in \mathcal{Y}$, then $|S_{y,y'}|$ will be small. Furthermore, we expect output labels which occur infrequently in the training data to be more difficult for the model to predict correctly, so $|S_{y,y'}|$ will scale with the easiness of $S_{y,y'}$ rather than difficulty, which is exactly the opposite of what we'd like.

2 Alternating minimization alternatives to objective 1

We'd like to alter objective function 1 such that the final cost weights do not reflect irrelevant properties of the data, but instead satisfy property 6 for some measure of prediction class difficulty D_S. One approach to this problem is to try to tease out the irrelevant biases from $|S|$ that objective function 1 incorporates into the cost weights. But $|S|$ depends on the predictions of the model, and these predictions might reflect the

[2] The model intuitively and empirically has a tendency toward this, but I haven't proven that it always holds, and I'm not sure that it does.

irrelevant biases. For example, for $S_{y,y'} \in \mathcal{S}_{[\mathcal{Y}]^2}$, $|S_{y,y'}|$ is determined by how often the model predicts y', and the frequency of this prediction depends partially on the frequency of y' in the training data (which is irrelevant to difficulty of $S_{y,y'}$). It seems hard to remove this irrelevant bias if the predictions are changing, so one idea would be to hold the predictions constant while adjusting the cost weights $\mathbf{v}$. This suggests a procedure where we alternate between shifting $\mathbf{w}$ alone to change the predictions and shifting $\mathbf{v}$ alone to reflect the difficulty of certain prediction classes under fixed $\mathbf{w}$. For example, we can shift $\mathbf{w}$ to take steps toward minimizing objective 1, and then hold $\mathbf{w}$ constant while shifting $\mathbf{v}$ independently of objective 1 to reflect the difficulty of various prediction classes under the fixed $\mathbf{w}$.

Under this alternating approach, for a fixed $\mathbf{w}$, we can adjust $\mathbf{v}$ to satisfy property 6 for a given difficulty measure $\mathbf{D}(M_\mathbf{w})$ using several methods including:

1. **Minimize Dot Product**: $\min_{\mathbf{v}\geq 0, \|\mathbf{v}\|_1=1} \mathbf{v}^\top \mathbf{D}(M_\mathbf{w})$
2. **Minimize Cosine Similarity**: $\min_{\mathbf{v}\geq 0, \|\mathbf{v}\|_1=1} \frac{\mathbf{v}^\top \mathbf{D}(M_\mathbf{w})}{|\mathbf{v}||\mathbf{D}(M_\mathbf{w})|}$
3. **'Inverted Distribution'**[3]: For $D_S(M_\mathbf{w}) \in [0,1]$, $v_s = \frac{1-D_S(M_\mathbf{w})}{|\mathcal{S}|-1}$

Of the above options, 1 and 2 prefer more sparse $\mathbf{v}$ whereas 3 ensures that $1 - v_S \propto D_S(M_\mathbf{w})$. For each of these methods of choosing $\mathbf{v}$, there are several measures of difficulty we could use as $\mathbf{D}(M_\mathbf{w})$:

1. κ **Difficulty**: [4] $D_S(M_\mathbf{w}) = \frac{P((\mathbf{x}_i,\mathbf{y}_i,M_\mathbf{w}(\mathbf{x}_i))\in S)-P(B)}{1-P(B_S)}$ where $P(B_S)$ is a base rate at which $(\mathbf{x}_i, \mathbf{y}_i, M_\mathbf{w}(\mathbf{x}_i))$ is expected to be in S. For example, with $\mathcal{S}_{[\mathcal{Y}]^2}$, we can choose $P(B_{S_{y,y'}}) = P(y_i = y)P(M_\mathbf{w}(\mathbf{x}_i) = y') + P(y_i = y')P(M_\mathbf{w}(\mathbf{x}_i) = y)$. This gives a measure of how much more often the model makes a prediction error in S than you'd expect if the model made predictions according to its output distribution independently of the input features. Intuitively, this measures how poorly how prone the model's feature weights are to prediction errors of type S independent of its label bias.
2. **Non-generalizability**: $D_S(M_\mathbf{w})$ is set to the difference between the training set error and dev error.
3. **Perceptron Loss per Example**:

$$D_S(M_\mathbf{w}) = \frac{\sum_{i=1}^{N} \mathbf{1}((\mathbf{x}_i, \mathbf{y}_i, M_\mathbf{w}(\mathbf{x}_i)) \in S)(\mathbf{w}^\top \mathbf{g}(\mathbf{x}_i, M_\mathbf{w}(\mathbf{x}_i)) - \mathbf{w}^\top \mathbf{g}(\mathbf{x}_i, \mathbf{y}_i))}{|S|}$$

We can gather other notions of 'difficulty' from learning theory (e.g. see `http://en.wikipedia.org/wiki/Stability_(learning_theory)` or from existing literature on SVMs (e.g. try something based on linear separability defined in `http://jmlr.org/papers/volume2/crammer01a/crammer01a.pdf`).

We have implemented a version of the cost learner that minimizes cosine similarity with the κ difficulty (minimization option 2 with difficulty option 1 above). This version performs better with $\mathcal{S}_{[\mathcal{Y}]^2}$ on data sets with non-uniform label distributions than the learner that just minimizes objective 1. Nevertheless, this new model is difficult to analyse and explain, and all of these alternating minimization ideas feel dirty and hackish. Instead, the idea proposed below is preferable if it works.

[3] Do people do anything like this in other places? Is there a name for this?

[4] This has the same form as Cohen's κ.

3 A better solution to issues with objective 1

Recall that objective function 1 uses difficulty measure $D_S(M) = |S_M|$, and that this is an issue because $|S_M|$ is biased by properties of the input data that are not representative of the difficulty of S. In at least some cases, the model's failure to account for this irrelevant bias is represented by the fact that the maximum value of $|S|$ varies across S, but objective function 1 treats each S the same way regardless of the varying maxima. For example, for $\mathcal{S}_{[\mathcal{Y}]^2}$, we don't expect $|S_{y,y'}|$ to be larger than $2NP(y_i = y)P(y_i = y')$ if the model is biased to predict y and y' at nearly the same frequency that they occur in the training data. This suggests that we might fix objective function 1 by modifying it so that the following property holds for the best cost weights:

$$\forall S, S' \in \mathcal{S}, v_S \leq v_{S'} \iff \frac{|S|}{n_S} \geq \frac{|S'|}{n_{S'}} \tag{8}$$

Where n_S is the maximum number of predictions we expect the model to make on prediction class S, and the difficulty of S is $D_S = \frac{|S|}{n_S}$ A further natural property of the model that follows with this would be:

$$\forall S \in \mathcal{S}, |S| > n_S \Rightarrow v_S = 0 \tag{9}$$

This property suggests that if the model makes even more incorrect predictions of class S than the expected maximum for that class, then the model should pay no cost for such predictions.

An alternative to objective function 1 that has these properties is:

$$\begin{aligned} &\min_{\mathbf{v} \geq 0, \mathbf{w}} \lambda_2 \|\mathbf{w}\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \\ &\quad + \sum_{i=1}^{N} \left(-\mathbf{w}^\top \mathbf{g}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{b}^\top \mathbf{l}(\mathbf{y}_i) + \max_{\mathbf{y} \in \mathcal{Y}_{x_i}} \left(\mathbf{w}^\top \mathbf{g}(\mathbf{x}_i, \mathbf{y}) + \mathbf{b}^\top \mathbf{l}(\mathbf{y}) + \mathbf{v}^\top \mathbf{c}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}) \right) \right) \\ &\quad - K\mathbf{v}^\top \mathbf{n} + \frac{K}{2} \|\mathbf{v}\|_{\mathbf{n}}^2 \end{aligned} \tag{10}$$

The intuition for this objective is that the $-K\mathbf{v}^\top \mathbf{n}$ term selects which S should have non-zero costs, and the $\frac{K}{2}\|\mathbf{v}\|_{\mathbf{n}}^2$ term orders the non-zero cost sets by their relative difficulties.[5]

The objective will be minimized with $v_S = 0$ on the boundary of $\mathbf{v} \geq 0$, or at either a critical or non-differentiable point. If the objective is minimized at a critical point, then:

$$\begin{aligned} &\frac{\partial}{\partial v_S} = K \sum_{i=1}^{N} \mathbf{1}((\mathbf{x}_i, \mathbf{y}_i, M(\mathbf{x}_i)) \in S) + Kn_s v_s - Kn_s = 0 \\ &\Rightarrow v_S = 1 - \frac{|S_M|}{n_S} \end{aligned} \tag{11}$$

So, ignoring the possibility of non-differentiable minima, the objective is minimized at:

$$v_S = \max\left(0, 1 - \frac{|S_M|}{n_S}\right) \tag{12}$$

Note that this value for v_S satisfies all of the desired properties discussed above. The non-differentiable minima are more difficult to think about, but their values for v_S also satisfy the above properties as a result of the following observations (which I'll try to turn into a proof if I have time and it seems useful):

[5] This is partially inspired by the objective function used in `http://papers.nips.cc/paper/3923-self-paced-learning-for-latent-variable-models.pdf`.

1. Let $\mathbf{Y}^* = \{(\mathbf{y}_1^*, \ldots, \mathbf{y}_N^*)\}$ be a a set of vectors of output labels for which the $\max$ operator in the objective is maximized at the objective's minimum. Notice that for any sequence of outputs $\mathbf{y}^* \in \mathbf{Y}^*$, there is a corresponding differentiable convex version of the objective function $o_{\mathbf{y}^*}$ with the $\max$ operator removed.

2. All non-differentiable points of the objective function are at intersections of the $o_{\mathbf{y}^*}$ functions.

3. If a minimum of the objective function is at an intersection of $o_{\mathbf{y}_1^*}$ and $o_{\mathbf{y}_2^*}$ that is non-differentiable in dimension v_S, then at this point, $\frac{\partial o_{\mathbf{y}_1^*}}{\partial v_S} = -\frac{\partial o_{\mathbf{y}_2^*}}{\partial v_S}$.

4. Suppose convex functions f_1 and f_2 have unique minima. Then, if $\mathbf{x}$ is a point such that $f_1(\mathbf{x}) = f_2(\mathbf{x})$ and $\frac{\partial f_1}{\partial x_i} = -\frac{\partial f_2}{\partial x_i}$, then x_i is between the minima of f_1 and f_2.

These observations suggest that if the objective is minimized at a non-differentiable point, then v_S at that point is between minima of the differentiable objectives of several tied, equally good, best label sequences. The values of the cost weights at the minima of the differentiable objectives are given by equation 12. So, the value of v_S at the non-differentiable minimum characterizes the difficulty of resolving S as a compromise between the difficulty estimates corresponding to each tied best label sequence.

3.1 Possible values for n_S

I've implemented several choices for n_S for $S \in \mathcal{S}_{[\mathcal{Y}]^2}$:

1. **None**: $n_S = 1$. So cost weights are non-normalized, and so the model should perform similarly to the original model that we started with.

2. **Logical**: $n_{\{y,y'\}}$ is the maximum over the number of times y occurs in the training data and the number of times y' occurs in the training data.

3. **Expected**: $n_{\{y,y'\}}$ is the expected number of times a model makes a mistake in $S_{y,y'}$ if the model predicts labels randomly according to their distribution in the training data.

I can write these up more formally later, but these are the general ideas for each choice of n_S. I've also implemented the corresponding choices of n_S for $S \in \mathcal{S}_{\mathcal{Y}^2}$ (ordered label costs).

Another interesting idea would be to choose n_S according to a baseline model over which we'd like to improve... or something like that...

4 Hyperparameter K is unnecessary

There is a short proof that it is unnecessarcy to perform a grid search search cost scalar K in the SVM and cost learning if you perform a wide enough search for the regularization hyper-parameters instead. In particular, for every value of K there is a corresponding set of regularization hyper-parameters that gives the same set of predictions, but scaled weights. Ask Jesse Dodge (`dodgejesse@gmail.com`) for a document showing the proof, or just try to figure it out—it's just a couple steps.