Learning to Rank

Karl Stratos

1 The Ranking Problem

Let $\mathcal{X}$ denote the query space and $[m] := \{1 \ldots m\}$ the finite target space. Let $\mathbf{perm}(m)$ denote the set of $m!$ permutations of $[m]$. The ranking problem is to learn a model $\Pi_\theta(x) \approx \Pi^*(x)$ where $\Pi^*(x) \subset \mathbf{perm}(m)$ is the set of gold rankings for each $x \in \mathcal{X}$. The model can be parameterized as an input-target score function $s_\theta(x, y) \in \mathbb{R}$ from which rankings can be extracted as $\Pi_\theta(x) = \{\pi \in \mathbf{perm}(m) : s_\theta(x, \pi(1)) \geq \cdots \geq s_\theta(x, \pi(m))\}$. For annotation, it is not practical to manually sort all m targets per query, thus we typically assume partial "pointwise" annotation: humans assign gold scores $s^*(x, y) \geq 0$ for some query-target pairs x, y. The score can be graded (e.g., 0–4 where 0 is least relevant and 4 is highly relevant) or binary (e.g., 0 if irrelevant and 1 if relevant).

Evaluation. Pick $K \leq m$ and let $\hat{y}_{1:K} = (\hat{y}_1 \ldots \hat{y}_K) \in [m]^K$ denote the top-K candidates under the model for x.[1] If the gold scores are binary $s^*(x, y) \in \{0, 1\}$, they can be viewed as an unordered set of relevant targets $\mathcal{Y}(x) = \{y \in [m] : s^*(x, y) = 1\}$. In this setting, frequently used metrics are (in $[0, 1]$; higher is better)

$$\textbf{Recall}(\mathcal{Y}(x), \hat{y}_{1:K}) = \frac{|\mathcal{Y}(x) \cap \{\hat{y}_1 \ldots \hat{y}_K\}|}{|\mathcal{Y}(x)|} \qquad \textbf{(recall at } K\textbf{)}$$

$$\textbf{MRR}(\mathcal{Y}(x), \hat{y}_{1:K}) = \frac{1}{\min_{k \in \{1 \ldots K\} :\, \hat{y}_k \in \mathcal{Y}(x)} k} \qquad \textbf{(mean reciprocal rank at } K\textbf{)} \tag{1}$$

$$\textbf{AP}(\mathcal{Y}(x), \hat{y}_{1:K}) = \frac{1}{|\mathcal{Y}(x)|} \sum_{k=1 : y_k \in \mathcal{Y}(x)}^{K} \frac{|\mathcal{Y}(x) \cap \{\hat{y}_1 \ldots \hat{y}_k\}|}{k} \qquad \textbf{(average precision at } K\textbf{)} \tag{2}$$

In (1), we assume the denominator is infinity if there is no relevant target in $\hat{y}_{1:K}$ (thus MRR is zero). Unlike recall, MRR is sensitive to the position of the captured relevant targets (taking a value in $\left\{1, \frac{1}{2}, \ldots, \frac{1}{K}, 0\right\}$), but does not care about lower-ranked relevant targets, so it is not suitable if $|\mathcal{Y}(x)| > 1$. AP is sensitive to the positions of multiple captured relevant targets.[2] For general (i.e., non-binary) gold scores $s^*(x, y) \in \mathbb{R}_{\geq 0}$, a standard metric is (also in $[0, 1]$)

$$\textbf{NDCG}(s^*, x, \hat{y}_{1:K}) = \left(\sum_{j=1}^{K} \frac{s^*(x, y_j^*)}{\log_2(j+1)} \right)^{-1} \sum_{k=1}^{K} \frac{s^*(x, \hat{y}_k)}{\log_2(k+1)} \qquad \textbf{(normalized discounted cumulative gain at } K\textbf{)} \tag{3}$$

where $y_1^* \ldots y_K^*$ are top-K gold targets under s^*. The log in the discount factor has the effect of diminishing the contribution of lower ranked relevant targets less severely.

2 Learning

Pointwise annotation implies "listwise" and "pairwise" annotation. For instance,

[1] We assume that $\hat{y}_{1:K}$ is unique per query x. This disregards the problem of spurious ambiguity (n targets with the same score generate $n!$ equally valid rankings), but it becomes negligible for any nontrivial model/data in practice.

[2] AP is the area under the precision-recall curve (thus a value in $[0, 1]$): $\sum_{k=1}^{K} \mathrm{P@}k \times \Delta\mathrm{R@}k$ where $\mathrm{P@}k$ is the precision at k and $\Delta\mathrm{R@}k = \frac{[[\hat{y}_k \in \mathcal{Y}(x)]]}{|\mathcal{Y}(x)|}$ is the change in recall. If $K = |\mathcal{Y}(x)|$, AP is maximized to 1 iff all relavant targets are ranked at the top; in this case, it is sometimes called label ranking average precision (LRAP).

(pointwise)	(listwise)	(pairwise)		
$s^*(x,1)=0$		$2 \triangleright 5 \mid x$	$5 \triangleright 1 \mid x$	$4 \triangleright 1 \mid x$
$s^*(x,2)=3$		$2 \triangleright 4 \mid x$	$5 \triangleright 3 \mid x$	$4 \triangleright 3 \mid x$
$s^*(x,3)=0$	$2 \triangleright 5 \equiv 4 \triangleright 1 \equiv 3 \mid x$	$2 \triangleright 1 \mid x$		
$s^*(x,4)=1$		$2 \triangleright 3 \mid x$		
$s^*(x,5)=1$				

Note that pairwise comparisons only consider targets with different scores. We will focus on learning from pairwise comparisons. We assume a training set of queries $\mathcal{S} \subset \mathcal{X}$ where each query $x \in \mathcal{S}$ is associated with N_x distinct target pairs $y_i \neq y'_i$ such that $y_i \triangleright y'_i \mid x$ and their weights $w_{x,i} \geq 0$:

$$\mathcal{S}(x) = \left\{ (w_{x,1}, y_1, y'_1) \dots (w_{x,N_x}, y_{N_x}, y'_{N_x}) \right\}$$

2.1 Logistic regression

We use (weighted) logistic regression to train $s_\theta : \mathcal{X} \times [m] \to \mathbb{R}$:[3]

$$J(\theta) = -\sum_{x \in S} \sum_{i=1}^{N_x} w_{x,i} \times \log \sigma(s_\theta(x, y_i) - s_\theta(x, y'_i)) \tag{4}$$

Instead of making $O(|S|\, m^2)$ backward calls to compute the gradient of J, we can develop a loss-specific precomputation scheme that makes $O(|S|\, m)$ backward calls. Let $J_{x,i}(\theta) = -w_{x,i} \times \log \sigma(s_\theta(x, y_i) - s_\theta(x, y'_i))$ denote the per-instance loss and define an instance-level reward

$$\lambda_{x,i}(\theta) := -w_{x,i} \times \sigma(s_\theta(x, y'_i) - s_\theta(x, y_i)) \tag{5}$$

which close to 0 if the model is correct and $-w_{x,i}$ if incorrect. It is easy to verify that $\lambda_{x,i}(\theta) = \frac{\partial J_{x,i}(\theta)}{\partial s_\theta(x,y_i)} = -\frac{\partial J_{x,i}(\theta)}{\partial s_\theta(x,y'_i)}$. We can also define a target-level adjustment

$$\lambda_{x,y}(\theta) := \sum_{i=1:\ y_i = y}^{N_x} \lambda_{x,i}(\theta) - \sum_{i=1:\ y'_i = y}^{N_x} \lambda_{i,x}(\theta) \tag{6}$$

which is positive if $s_\theta(x, y)$ is too large, negative if $s_\theta(x, y)$ is too small, and 0 if neither. We can write

$$\begin{aligned} \nabla J(\theta) &= \sum_{x \in S} \sum_{i=1}^{N_x} \left(\lambda_{x,i}(\theta) \nabla s_\theta(x, y_i) - \lambda_{x,i}(\theta) \nabla s_\theta(x, y'_i) \right) & (7) \\ &= \sum_{x \in S} \sum_{y \in [m]} \lambda_{x,y}(\theta) \nabla s_\theta(x, y) & (8) \end{aligned}$$

(7) is the chain rule. (8) uses the fact that each $y \in [m]$ is either absent or exactly one of y_i or y'_i in each instance.

2.2 Boosting

We use tree boosting to estimate $f : \mathbb{R}^d \to \mathbb{R}$ that minimizes

$$L(f) = -\sum_{x \in S} \sum_{i=1}^{N_x} w_{x,i} \times \log \sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i)) \Big) \tag{9}$$

[3]Assuming universality and constant (nonzero) weights, we will have $s_{\theta^*}(x, y) - s_{\theta^*}(x, y') = \log(\frac{p}{1-p})$ where p is the population probability that $y \triangleright y' \mid x$ given random x, y, y' for an optimal θ^*.

where $\phi(x, y) \in \mathbb{R}^d$ is a feature function. It is the same objective as (4), just re-expressed as a functional. Define for all $f : \mathbb{R}^d \to \mathbb{R}$ and $x \in \mathcal{S}$.

$$\lambda_{x,i}(f) := -w_{x,i} \times \sigma(f(\phi(x, y_i')) - f(\phi(x, y_i))) \qquad \forall i \in [N_x]$$

$$\lambda_{x,y}(f) := \sum_{i=1:\, y_i = y}^{N_x} \lambda_{x,i}(f) - \sum_{i=1:\, y_i' = y}^{N_x} \lambda_{i,x}(f) \qquad \forall y \in [n]$$

$$\rho_{x,y}(f) := -\sum_{i=1:\, y_i = y}^{N_x} \lambda_{x,i}(f)\sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y_i'))\Big) - \sum_{i=1:\, y_i' = y}^{N_x} \lambda_{x,i}(f)\sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y_i'))\Big) \qquad \forall y \in [n]$$

The first two definitions are analogous to (5) and (6). Let $\mathcal{H}_{\text{tree}}(d)$ is the set of regression trees $h : \mathbb{R}^d \to \mathbb{R}$, each characterized by leaf nodes $\mathbf{Leaf}_1 \dots \mathbf{Leaf}_R \subset \mathbb{R}^d$. See Appendix A.4 for a derivation of the algorithm below.

```
TreeBoosting
Input: labeled queries $\mathcal{S} \subset \mathcal{X}$, number of boosting rounds $T$
Output: local minimum $f^{(T)} : \mathbb{R}^d \to \mathbb{R}$ of (9)
  1. $f^{(0)} \leftarrow 0$
  2. For $t = 1 \dots T$, compute
       $\{\mathbf{Leaf}_r^{(t)}\}_{r=1}^{R_t} \leftarrow \arg\min_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{x \in \mathcal{S}} \sum_{y \in [n]} \rho_{x,y}(f^{(t-1)}) \left(-\frac{\lambda_{x,y}(f^{(t-1)})}{\rho_{x,y}(f^{(t-1)})} - h(\phi(x, y))\right)^2$
       $\hat{\gamma}_r^{(t)} \leftarrow -\frac{\sum_{x \in \mathcal{S}} \sum_{y \in [n]:\, \phi(x,y) \in \mathbf{Leaf}_r^{(t)}} \lambda_{x,y}(f^{(t-1)})}{\sum_{x \in \mathcal{S}} \sum_{y \in [n]:\, \phi(x,y) \in \mathbf{Leaf}_r^{(t)}} \rho_{x,y}(f^{(t-1)})}$    $\forall r = 1 \dots R_t$
       $f^{(t)}(\phi(x, y)) \leftarrow f^{(t-1)}(\phi(x, y)) + \sum_{r=1}^{R_t} \hat{\gamma}_r^{(t)} [[\phi(x, y) \in \mathbf{Leaf}_r^{(t)}]]$    $\forall x \in \mathcal{X},\ y \in [n]$
  3. Return $f^{(T)} \in \text{span}(\mathcal{H}_{\text{tree}}(d))$.
```

2.3 Incorporating Listwise Features

LambdaRank [1] and LambdaMART [13] are the logistic regressor (4) and boosted trees (9) with the choice of weight, for each (x, y_i, y_i'),

$$w_{x,i} = \Big|\mathbf{NDCG}(s^*, x, \hat{y}_{1:K}) - \mathbf{NDCG}(s^*, x, \texttt{swap}(\hat{y}_{1:K}, y_i, y_i'))\Big| \tag{10}$$

$\hat{y}_{1:K}$ is the top-K targets under the model and $\texttt{swap}(\hat{y}_{1:K}, a, b)$ is the result of swapping a, b. Thus a target pair that substantially changes the current NDCG score when swapped will receive a large weight; a target pair that makes no change in NDCG will be ignored.

Optimization. Since $w_{x,i}$ is now a complicated function of the model parameter (e.g., nonsmooth; NDCG does not change for most continuous changes in the parameter), directly optimizing the objective is no longer feasible. An empirically successful scheme is the following: in each optimization step for a labeled query $x \in \mathcal{S}$:

1. Rank the all targets $[m]$ using the current model $f(x, y) \in \mathbb{R}$, calculate the NDCG-based weights $w_{x,i}$ (10).

2. Treat $w_{x,i}$ as constant and update the model parameter to minimize $-\sum_x \sum_i w_{x,i} \log \sigma(f(x, y_i) - f(x, y_i'))$.

After each step, we "refresh" the weights $w_{x,i}$ to use the new NDCG score under the updated model. This is not guaranteed to decrease the loss (e.g., the loss may shoot up when we refresh the weights). However, if we assume that pairwise ranking is correlated with the change in NDCG under a pairwise swap (which is reasonable), then refreshing $w_{x,i}$ will further decrease the loss. In this case we can view this as an alternating optimization algorithm.

3 Discussions

3.1 Losses

Ranking losses typically take one of the three forms: pointwise, pairwise, and listwise.

$$\begin{aligned}
J_{\text{point}}(\theta) &= \sum_{(x,y,z)} l_{\text{point}}(s_\theta(x,y), z) && x \in \mathcal{X},\ y \in [m],\ z \in \mathbb{R} \\
J_{\text{pair}}(\theta) &= \sum_{(x,y,y')} l_{\text{pair}}(s_\theta(x,y) - s_\theta(x,y')) && x \in \mathcal{X},\ y, y' \in [m] : y \rhd y' \mid x \\
J_{\text{list}}(\theta) &= \sum_{(x,y_1 \ldots y_K, z_1 \ldots z_K)} l_{\text{list}}(s_\theta(x,y_1) \ldots s_\theta(x,y_K), z_1 \ldots z_K) && x \in \mathcal{X},\ y_1 \ldots y_K \in [m],\ z_1 \ldots z_K \in \mathbb{R}
\end{aligned}$$

The pointwise loss can be either regression (e.g., $l_{\text{point}}(t,z) = ||z - t||^2$) or classification if the score is binary (e.g., $l_{\text{point}}(t,z) = -zt + \log(1 + e^t)$ for $z \in \{0,1\}$). The pairwise loss can be any upper bound on the zero-one loss $[[t \geq 0]]$ (e.g., the hinge loss $l_{\text{pair}}(t) = \max(0, 1-t)$, aka., "RankSVM" [6], or the logistic loss $l_{\text{pair}}(t) = -\log \sigma(t)$). Lambda{Rank,MART} uses the pairwise logistic loss but (heuristically) weighs it by the change in NDCG. A simple listwise loss is the cross entropy loss:

$$l_{\text{list}}(t_1 \ldots t_K, z_1 \ldots z_K) = -\sum_{k=1}^{K} p_k(z_1 \ldots z_K) \log \left(\frac{\exp(t_k)}{\sum_{k'=1}^{K} \exp(t_{k'})} \right)$$

where $p_k(u) \in \Delta^{K-1}$ is a gold distribution over the K targets obtained by normalizing $z_1 \ldots z_K$ (e.g., $p_k(u) \propto e^{u_k}$, aka., ListNet [2], or just $p_k(u) \propto u_k$ since $u \geq 0$ [10]). Other listwise losses aim to capture the ranking component more directly. For instance, assuming WLOG that $y_1 \rhd \cdots \rhd y_K \mid x$, we can consider the likelihood loss

$$l_{\text{list}}(t_1 \ldots t_K, z_1 \ldots z_K) = -\sum_{k=1}^{K} \alpha(k) \times \log \left(\frac{\exp(t_k)}{\sum_{k'=k}^{K} \exp(t_{k'})} \right)$$

where we use the chain rule to decompose the likelihood (ListMLE [14]); $\alpha(k)$ is a decreasing function (e.g., $\alpha(k) = 2^{K-k} - 1$) proposed in [8] to emphasize the top targets (p-ListMLE). There is a body of work on relating listwise losses to ranking metrics (e.g., an optimal ListNet model does not imply the optimal NDCG score [11]) and developing their differentiable approximations [9].

Pairwise comparisons. We have assumed that pairwise comparisons are extracted from pointwise scores "for free". But there are often situations where pointwise scores are difficult to obtain while pairwise comparisons are easy (e.g., the human perception of quality for the task is too subjective/subtle, but it is still relatively easy to tell which of the two given inputs is "better"). Pairwise losses are suitable in these situations and have been used for training a reward model in the context of reinforcement learning [12]. There is also a line of theoretical works that relate pairwise comparisons with ranking. In general, we must make $O(m \log m)$ comparisons to recover the underlying ranking of m targets (e.g., binary sort) because each comparison provides 1 bit of information about $m!$ rankings ($\log(m!) = O(m \log m)$). It is possible to reduce the number of comparisons by active learning under certain assumptions (e.g., [7]).

3.2 Web Search

Ranking has been addressed largely for web search where a target $y \in [m]$ is an indexed online document. Each query-document pair (x, y) is represented as a feature vector $\phi(x, y) \in \mathbb{R}^d$ that encodes their text match information (e.g., BM25, proximity), text and meta-information statistics (e.g., term frequency, number of clicks), web graph (e.g., PageRank) and time (freshness of y) information, and many other types of information and statistics. An influential dataset is the Yahoo! Learning to Rank Challenge (2010) dataset, which consists of 36k (randomly sampled, thus non-unique) Yahoo! queries and 883k documents [3]. Each query is associated with a varying number of documents (24 on average, but as large as 100), extracted from top results of multiple search engines and manually labeled with 5 relevance grades.

Boosted trees with pairwise losses were the most successful on the web search datasets in the early 2010 (e.g., a LambdaMART-based ensemble won the Yahoo! challenge). This inordinately skewed the research efforts on ranking to boosting and pairwise losses. However, recent works find that neural models are on par with or better than boosted trees given a suitable feature normalization, architecture, and loss. For instance, [10] use the feature transformation $u \mapsto \log(1+|u|)\mathbf{sign}\,(u)+\mathcal{N}(0,\sigma^2 I_d)$ and self-attention over targets in a listwise loss (cross entropy) and find that this is competitive with a strong implementation of LambdaMART (LightGBM). While tree-based learning is indeed more robust to feature scales, there is no intrinsic reason that ranking should be associated with boosted trees and pairwise losses (in fact, listwise losses work better for neural models [10]).

Current limitations. In the context of web search, the score $s_\theta(x, y) \in \mathbb{R}$ should measure how well the document y fulfills the purpose of the query x. As usual, the best approach for this task is probably large-scale deep learning directly from natural signals. But the progress in this direction is blocked because the public datasets only release anonymized information with "unnatural" numeric features. Rather than applying a model directly on the query and document (x, y) in natural language (plus meta-information), we are applying it to a bunch of statistics and consequently need to worry about normalizing their values. This is a primitive setting that favors simple classifiers and kills the benefits of the deep understanding capabilities of neural models from pretraining on natural inputs.

References

[1] Burges, C., Ragno, R., and Le, Q. (2006). Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, **19**.

[2] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136.

[3] Chapelle, O. and Chang, Y. (2011). Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, pages 1–24. PMLR.

[4] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, **28**(2), 337–407.

[5] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[6] Graepel, T., Obermayer, K., *et al.* (2000). Large margin rank boundaries for ordinal regression. In *Advances in large margin classifiers*, pages 115–132. the MIT Press.

[7] Jamieson, K. G. and Nowak, R. (2011). Active ranking using pairwise comparisons. *Advances in neural information processing systems*, **24**.

[8] Lan, Y., Zhu, Y., Guo, J., Niu, S., and Cheng, X. (2014). Position-aware listmle: A sequential learning process for ranking. In *UAI*, pages 449–458. Citeseer.

[9] Qin, T., Liu, T.-Y., and Li, H. (2010). A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, **13**(4), 375–397.

[10] Qin, Z., Yan, L., Zhuang, H., Tay, Y., Pasumarthi, R. K., Wang, X., Bendersky, M., and Najork, M. (2021). Are neural rankers still outperformed by gradient boosted decision trees? In *International Conference on Learning Representations*.

[11] Ravikumar, P., Tewari, A., and Yang, E. (2011). On ndcg consistency of listwise ranking methods. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 618–626. JMLR Workshop and Conference Proceedings.

[12] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. (2020). Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, **33**, 3008–3021.

[13] Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, **13**(3), 254–270.

[14] Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199.

A Boosting

We view the loss as a functional $L(f) = \mathbf{E}[l(f(X), Y)]$ where $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ is an input-label pair drawn iid from $\mathbf{pop}_{XY}$, $f(x) \in \mathbb{R}^m$ is any function over $x \in \mathcal{X}$, and $l : \mathbb{R}^m \times \mathcal{Y} \to \mathbb{R}$ is a per-example loss function. A general scheme for boosting is as follows, assuming some base hypothesis class $\mathcal{H}$ of mappings $h : \mathcal{X} \to \mathbb{R}^m$.

1. Pick an initial function $f^{(0)} \in \mathcal{H}$.
2. For $t = 1 \ldots T$:
   (a) Approximate the steepest descent direction of L at $f^{(t-1)}$,[4]
   $$h^{(t)} \approx \operatorname*{arg\,min}_{h \in \mathcal{H}} \; L(f^{(t-1)} + h) \tag{11}$$
   (b) Optimize the step size (i.e., line search),
   $$\eta_t = \operatorname*{arg\,min}_{\eta \in \mathbb{R}} \; L(f^{(t-1)} + \eta h^{(t)})$$
   (c) Set $f^{(t)} = f^{(t-1)} + \eta_t h^{(t)}$.
3. Return $f^{(T)} = f^{(0)} + \sum_{t=1}^{T} \eta_t h^{(t)} \in \operatorname{span}(\mathcal{H})$ as a local optimum of $L(f)$.

A central step is computing the descent direction (11). Let $\nabla l(z, y) \in \mathbb{R}^m$ and $\nabla^2 l(z, y) \in \mathbb{R}^{m \times m}$ denote the gradient and Hessian of l with respect to the input $z \in \mathbb{R}^m$ (we assume they exist).

Lemma A.1 (Second-order approximation)**.**

$$L(f^{(t-1)} + h) \approx \mathbf{E}\Big[\Big(-\nabla^2 l(f^{(t-1)}(X), Y)^{-1} \nabla l(f^{(t-1)}(X), Y) - h(X)\Big)^\top \nabla^2 l(f^{(t-1)}(X), Y) \\ \Big(-\nabla^2 l(f^{(t-1)}(X), Y)^{-1} \nabla l(f^{(t-1)}(X), Y) - h(X)\Big) \Big] \tag{12}$$

We may further simplify (12) using diagonal and identity approximations of the Hessian. Let $l'_j(z, y) = [\nabla l(z, y)]_j$ and $l''_j(z, y) = [\nabla^2 l(z, y)]_{j,j}$ denote the first- and second-order partial derivative of $l(z, y)$ with respect to $z_j \in \mathbb{R}$.

Corollary A.2 (Second-order diagonal approximation)**.**

$$L(f^{(t-1)} + h) \approx \mathbf{E}\left[\sum_{j=1}^{m} l''_j(f^{(t-1)}(X), Y) \left(-\frac{l'_j(f^{(t-1)}(X), Y)}{l''_j(f^{(t-1)}(X), Y)} - h_j(X) \right)^2 \right] \tag{13}$$

Corollary A.3 (First-order approximation)**.**

$$L(f^{(t-1)} + h) \approx \mathbf{E}\left[\sum_{j=1}^{m} \Big(-l'_j(f^{(t-1)}(X), Y) - h_j(X) \Big)^2 \right] \tag{14}$$

In practice we use the empirical functional $L(f) = \sum_{i=1}^{N} l(f(x_i), y_i)$ where $(x_1, y_1) \ldots (x_N, y_N) \sim \mathbf{pop}_{XY}$. Approximating the steepest descent direction by (14) and (13), we have

$$h^{(t)} = \operatorname*{arg\,min}_{h \in \mathcal{H}} \sum_{i=1}^{N} \sum_{j=1}^{m} \Big(-l'_j(f^{(t-1)}(x_i), y_i) - h_j(x_i) \Big)^2 \qquad \textbf{(gradient boosting)} \tag{15}$$

$$h^{(t)} = \operatorname*{arg\,min}_{h \in \mathcal{H}} \sum_{i=1}^{N} \sum_{j=1}^{m} l''_j(f^{(t-1)}(x_i), y_i) \left(-\frac{l'_j(f^{(t-1)}(x_i), y_i)}{l''_j(f^{(t-1)}(x_i), y_i)} - h_j(x_i) \right)^2 \qquad \textbf{(Newton boosting)} \tag{16}$$

Thus boosting can be performed for any loss function l, assuming the feasibility of (weighted) least squares for $\mathcal{H}$ and calculation of l'_j and l''_j on all examples.

[4] If can compute $h^* = \operatorname{arg\,min}_{h \in \mathcal{H}} \; L(f^{(t-1)} + h)$ exactly with a universal $\mathcal{H}$, we are done (i.e., $f^{(t-1)} + h^* = \operatorname{arg\,min}_{f : \mathcal{X} \to \mathbb{R}^m} L(f)$).

Loss function. The input-label pair in $l(f(x), y) \in \mathbb{R}$ are usually framed as

1. $f(x) \in \mathbb{R}^m$: scores of m "categories" for $x \in \mathcal{X}$ (e.g., $m = 1$ for regression, $m = K$ for multiclass classification)
2. $y \in \mathcal{Y}$: an instruction on how the loss should be calculated based on these scores

The hypothesis $h(x) \in \mathbb{R}^m$ is viewed as predicting the "deltas" for these m categories. For instance, in gradient boosting, $h_j(x) \in \mathbb{R}$ is trained to predict the negative j-th partial gradient

$$-l'_j(f(x), y) = -\frac{\partial l((f_1(x), \ldots, f_m(x)), y)}{\partial f_j(x)}$$

which is the sensitivity of l with respect to the score of the j-th category (called "pseudoresponse"). This becomes the residual with the squared loss and the probability gap with the logistic loss.

A.1 Boosted Trees

We often use regression trees as the base hypothesis class because they are easy to optimize for weighted least squares. More specifically, let $\mathcal{H}_{\text{tree}}(d)$ denote the set of (scalar-valued) regression trees $h : \mathbb{R}^d \to \mathbb{R}$. We assume that estimating

$$h^* = \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^{N} w_i \left(r_i - h(x_i)\right)^2 = \{\mathbf{Leaf}_r, b_r\}_{r=1}^{R}$$

is easy for any response variables $r_1 \ldots r_N \in \mathbb{R}$ and weights $w_1 \ldots w_N \in \mathbb{R}$. Here, $\mathbf{Leaf}_1 \ldots \mathbf{Leaf}_R \subset \mathbb{R}^d$ are leaf nodes (a partition of $\mathbb{R}^d$) and $b_1 \ldots b_R \in \mathbb{R}$ are the leaf values that characterize h^*: $h^*(x) = \sum_{r=1}^{R} b_r\, [[x \in \mathbf{Leaf}_r]]$. We will cover three common settings in the loss function $l : \mathbb{R}^m \times \mathcal{Y} \to \mathbb{R}$: (1) $m = 1$, (2) $m > 1$, and (3) $m > 1$ with parameter sharing.

A.2 $m = 1$

The goal is to optimize a scalar-valued predictor for a scalar-valued loss

$$\boxed{f^* = \operatorname*{arg\,min}_{f:\mathbb{R}^d \to \mathbb{R}} \sum_{i=1}^{N} l(f(x_i), y) \qquad\qquad l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}}$$

(16) is a single regression tree

$$\{\mathbf{Leaf}_r, b_r\}_{r=1}^{R} = \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^{N} l''(f(x_i), y_i) \left(-\frac{l'(f(x_i), y_i)}{l''(f(x_i), y_i)} - h(x_i)\right)^2$$

Now, instead of doing the line search

$$\eta_t = \operatorname*{arg\,min}_{\eta \in \mathbb{R}} \sum_{i=1}^{N} l\left(f(x_i) + \eta \sum_{r=1}^{R} b_r\, [[x_i \in \mathbf{Leaf}_r]]\,, y_i\right)$$

and making the update $\bar{f}(x) = f(x) + \eta_t \sum_{r=1}^{R} b_r\, [[x \in \mathbf{Leaf}_r]]$, it is common to re-optimize the leaf values along with the step size:

$$\gamma^* = \operatorname*{arg\,min}_{\gamma \in \mathbb{R}^R} \sum_{i=1}^{N} l\left(f(x_i) + \sum_{r=1}^{R} \gamma_r\, [[x_i \in \mathbf{Leaf}_r]]\,, y_i\right) \tag{17}$$

whereupon the new update is $\bar{f}(x) = f(x) + \sum_{r=1}^{R} \gamma_r^*\, [[x_i \in \mathbf{Leaf}_r]]$. Since the leaf nodes partition $\mathbb{R}^d$, we can estimate each value γ_r^* separately only considering examples that belong to $\mathbf{Leaf}_r$:

$$\gamma_r^* = \operatorname*{arg\,min}_{\gamma \in \mathbb{R}} \sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^{N} l\left(f(x_i) + \gamma, y_i\right) \approx -\frac{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^{N} l'(f(x_i), y_i)}{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^{N} l''(f(x_i), y_i)} =: \hat{\gamma}_r$$

where the approximation is given by a single Newton step from $\gamma = 0$. After T iterations, the final predictor has the form $f^{(T)}(x) = f^{(0)}(x) + \sum_{t=1}^{T} \hat{\gamma}_r^{(t)} [[x \in \mathbf{Leaf}_r^{(t)}]] \in \mathbb{R}$ where $\mathbf{Leaf}_1^{(t)} \ldots \mathbf{Leaf}_{R_t}^{(t)}$ are the leaf nodes of the t-th tree and $\hat{\gamma}_1^{(t)} \ldots \hat{\gamma}_{R_t}^{(t)}$ are the re-optimized values.

Example: least squares. $x \in \mathbb{R}^d$, $y \in \mathbb{R}$

$$\begin{aligned} l(f(x), y) &:= (y - f(x))^2 \\ l'(f(x), y) &= -2(y - f(x)) \\ l''(f(x), y) &= 2 \end{aligned}$$

Boosting step:

$$\begin{aligned} \{\mathbf{Leaf}_r, b_r\}_{r=1}^R &= \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^N ((y_i - f(x_i)) - h(x_i))^2 \\ \hat{\gamma}_r &= \sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^N (y_i - f(x_i)) && \forall r = 1 \ldots R \end{aligned}$$

Example: logistic regression. $x \in \mathbb{R}^d$, $y \in \{\pm 1\}$

$$\begin{aligned} l(f(x), y) &:= -\log \sigma(y f(x)) \\ l'(f(x), y) &= -y\sigma(-y f(x)) \\ l''(f(x), y) &= \sigma(-y f(x))\sigma(y f(x)) = |\hat{y}|\,(1 - |\hat{y}|) \end{aligned}$$

where $\hat{y} = y\sigma(-y f(x))$ is the pseudoresponse. Boosting step:

$$\begin{aligned} \{\mathbf{Leaf}_r, b_r\}_{r=1}^R &= \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^N |\hat{y}_i|\,(1 - |\hat{y}_i|) \left(\frac{\hat{y}_i}{|\hat{y}_i|\,(1 - |\hat{y}_i|)} - h(x_i) \right)^2 \\ \hat{\gamma}_r &= \frac{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^N \hat{y}_i}{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r}^N |\hat{y}_i|\,(1 - |\hat{y}_i|)} && \forall r = 1 \ldots R \end{aligned}$$

A.3 $m > 1$

The goal is to optimize a vector-valued predictor for a vector-valued loss

$$\boxed{f^* = \operatorname*{arg\,min}_{f : \mathbb{R}^d \to \mathbb{R}^m} \sum_{i=1}^N l(f(x_i), y) \qquad\qquad l : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}}$$

The hypothesis $h(x) = (h_1(x) \ldots h_m(x))$ uses regression trees $h_1 \ldots h_m \in \mathcal{H}_{\text{tree}}(d)$ to predict m scores for x. Since they are independent, we can reduce (16) to optimizing each tree individually per score j:

$$\{\mathbf{Leaf}_r^{(j)}, b_r^{(j)}\}_{r=1}^{R_j} = \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^N l_j''(f(x_i), y_i) \left(-\frac{l_j'(f(x_i), y_i)}{l_j''(f(x_i), y_i)} - h(x_i) \right)^2$$

Again, instead of a line search

$$\eta_t = \operatorname*{arg\,min}_{\eta \in \mathbb{R}} \sum_{i=1}^N l \left(f(x_i) + \eta \left(\sum_{r=1}^{R_j} b_r^{(j)} \left[\left[x_i \in \mathbf{Leaf}_r^{(j)} \right]\right] \right)_{j=1}^m, y_i \right)$$

we re-optimize the leaf values,

$$\{\gamma_*^{(j)}\}_{j=1}^m = \operatorname*{arg\,min}_{\gamma^{(j)} \in \mathbb{R}^{R_j}\ \forall j = 1 \ldots m} \sum_{i=1}^N l \left(f(x_i) + \left(\sum_{r=1}^{R_j} \gamma_r^{(j)} \left[\left[x_i \in \mathbf{Leaf}_r^{(j)} \right]\right] \right)_{j=1}^m, y_i \right)$$

Unlike (17), this objective does not factorize over $\gamma_r^{(j)}$ because each example i will trigger $\gamma_{r_1}^{(1)} \ldots \gamma_{r_m}^{(m)}$ for some $r_j \in [R_j]$ which may interact nonlinearly inside l. However, we can again derive a simple approximation based on a single Newton step from $\gamma_r^{(j)} = 0$ by using a diagonal approximation of the Hessian:

$$[\gamma_*^{(j)}]_r \approx -\frac{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r^{(j)}}^N l_j'(f(x_i), y_i)}{\sum_{i=1:\ x_i \in \mathbf{Leaf}_r^{(j)}}^N l_j''(f(x_i), y_i)} =: \hat{\gamma}_r^{(j)}$$

After T iterations, the final predictor has the form $f_j^{(T)}(x) = f_j^{(0)}(x) + \sum_{t=1}^T \sum_{r=1}^{R_{t,j}} \hat{\gamma}_r^{(t,j)} [[x \in \mathbf{Leaf}_r^{(t,j)}]] \in \mathbb{R}$ where $\mathbf{Leaf}_1^{(t,j)} \ldots \mathbf{Leaf}_{R_{t,j}}^{(t,j)}$ are the leaf nodes of the t-th iteration's j-th tree and $\hat{\gamma}_1^{(t,j)} \ldots \hat{\gamma}_{R_{t,j}}^{(t,j)}$ are the re-optimized values.

Example: multiclass classification. $x \in \mathbb{R}^d$, $y \in \{1 \dots m\}$

$$l(f(x), y) := -\log p(y|x; f) \qquad p(j|x; f) := \frac{e^{f_j(x)}}{\sum_{k=1}^m e^{f_k(x)}} \qquad \forall j = 1 \dots m$$

$$l'_j(f(x), y) = p(j|x; f) - [[j = y]]$$
$$l''_j(f(x), y) = p(j|x; f)(1 - p(j|x; f))$$

As in logistic regression, denoting the pseudoresponse by $\hat{y} = [[j = y]] - p(j|x; f)$ and using the fact that $|\hat{y}|(1 - |\hat{y}|) = p(j|x; f)(1 - p(j|x; f))$, we can write the boosting step in the nicer form:

$$\{\mathbf{Leaf}_r^{(j)}, b_r^{(j)}\}_{r=1}^{R_j} = \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^N |\hat{y}_i|(1 - |\hat{y}_i|) \left(\frac{\hat{y}_i}{|\hat{y}_i|(1 - |\hat{y}_i|)} - h(x_i) \right)^2 \qquad \forall j = 1 \dots m$$

$$\hat{\gamma}_r^{(j)} = \frac{\sum_{i=1:\, x_i \in \mathbf{Leaf}_r^{(j)}}^N \hat{y}_i}{\sum_{i=1:\, x_i \in \mathbf{Leaf}_r^{(j)}}^N |\hat{y}_i|(1 - |\hat{y}_i|)} \qquad \forall j = 1 \dots m, r = 1 \dots R_j$$

Historical notes: tree boosting for multiclass classification has been proposed in LogitBoost [4]. It makes the normalization assumption $\sum_{j=1}^m f_j(x) = 0$ to get rid of overparameterization so the updates are scaled by $\frac{m-1}{m}$. Gradient boosting has also been proposed (aka. "MART") [5], arguing that the second-order update is numerically unstable when $p(1 - p)$ is small. However, it does not seem to be a serious issue in practice and Newton boosting is preferred.

A.4 $m > 1$ with parameter sharing

The goal is to optimize a scalar-valued predictor for a vector-valued loss

$$\boxed{f^* = \operatorname*{arg\,min}_{f : \mathbb{R}^d \to \mathbb{R}} \sum_{i=1}^N l((f(\phi(x_i, k)))_{k=1}^m, y) \qquad l : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}}$$

The predictor is shared across the m "feature expansions" to yield m scores. Here, we assume a feature function $\phi : \mathcal{X} \times \mathcal{C} \to \mathbb{R}^d$ that yields a d-dimensional vector representation for each input-category pair. $\mathcal{C} \supset \{1 \dots m\}$ is the space of all possible categories. This formulation allows for zero-shot inference (i.e., we can apply f to an unseen catogery $k \in \mathcal{C} \backslash \{1 \dots m\}$ at test time). To make the notation tractable we will shorthand $f^\phi(x) := (f(\phi(x, k)))_{k=1}^m$.

The hypothesis takes the form $x \mapsto (h_{\text{shared}}(\phi(x, 1)) \dots h_{\text{shared}}(\phi(x, m)))$ where $h_{\text{shared}} \in \mathcal{H}_{\text{tree}}(d)$ is obtained by (16):

$$\{\mathbf{Leaf}_r, b_r\}_{r=1}^R = \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{i=1}^N \sum_{j=1}^m l''_j(f^\phi(x_i), y_i) \left(-\frac{l'_j(f^\phi(x_i), y_i)}{l''_j(f^\phi(x_i), y_i)} - h(x_i) \right)^2$$

(i.e., a single regression tree is multitasking m weighted least squares tasks). Again, instead of a line search

$$\eta_t = \operatorname*{arg\,min}_{\eta \in \mathbb{R}} \sum_{i=1}^N l \left(f^\phi(x_i) + \eta \left(\sum_{r=1}^R b_r \, [[\phi(x_i, k) \in \mathbf{Leaf}_r]] \right)_{k=1}^m, y_i \right)$$

we re-optimize the leaf values,

$$\gamma^* = \operatorname*{arg\,min}_{\gamma \in \mathbb{R}^R} \sum_{i=1}^N l \left(f^\phi(x_i) + \left(\sum_{r=1}^R \gamma_r \, [[\phi(x_i, k) \in \mathbf{Leaf}_r]] \right)_{k=1}^m, y_i \right)$$

Again, unlike (17), it does not factorize over γ_r because each example i can trigger multiple values of γ (e.g., $\phi(x_i, 1) \in \mathbf{Leaf}_r$ and $\phi(x_i, 2) \in \mathbf{Leaf}_{r'}$ will trigger $\gamma_r, \gamma_{r'}$), which may interact nonlinearly inside l. But again using a diagonal approximation of the Hessian, we can derive a simple approximation based on a Newton step from $\gamma = 0_R$:

$$\gamma_r^* \approx -\frac{\sum_{i=1}^N \sum_{j=1:\, \phi(x_i, j) \in \mathbf{Leaf}_r}^m l'_j(f^\phi(x_i), y_i)}{\sum_{i=1}^N \sum_{j=1:\, \phi(x_i, j) \in \mathbf{Leaf}_r}^m l''_j(f^\phi(x_i), y_i)} =: \hat{\gamma}_r$$

After T iterations, the final predictor can predict the score of any input $x \in \mathcal{X}$ and category $k \in \mathcal{C}$ by $f_k^{(T)}(x) = f^{(0)}(\phi(x, k)) + \sum_{t=1}^T \sum_{r=1}^{R_t} \hat{\gamma}_r^{(t)} [[\phi(x, k) \in \mathbf{Leaf}_r^{(t)}]] \in \mathbb{R}$ where $\mathbf{Leaf}_1^{(t)} \dots \mathbf{Leaf}_{R_t}^{(t)}$ are the leaf nodes of the t-th tree and $\hat{\gamma}_1^{(t)} \dots \hat{\gamma}_{R_t}^{(t)}$ are the re-optimized values.

Example: pairwise comparisons. $x \in \mathcal{X}$, $y = \{(w_{x,1}, y_1, y'_1) \dots (w_{x,N_x}, y_{N_x}, y'_{N_x})\}$ where $w_{x,i} \geq 0$, $y_i, y'_i \in \{1 \dots m\}$, and $y_i \neq y'_i$ for $i = 1 \dots N_x$

$$\begin{aligned}
l(f^\phi(x), y) &:= -\sum_{i=1}^{N_x} w_{x,i} \times \log \sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i))\Big) \\
l'_j(f^\phi(x), y) &= -\sum_{i=1:\ y_i=j}^{N_x} w_{x,i} \times \sigma\Big(f(\phi(x, y'_i)) - f(\phi(x, y_i))\Big) + \sum_{i=1:\ y'_i=j}^{N_x} w_{x,i} \times \sigma\Big(f(\phi(x, y'_i)) - f(\phi(x, y_i))\Big) \\
l''_j(f^\phi(x), y) &= \sum_{i=1:\ y_i=j}^{N_x} w_{x,i} \times \sigma\Big(f(\phi(x, y'_i)) - f(\phi(x, y_i))\Big)\sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i))\Big) + \\
&\quad \sum_{i=1:\ y'_i=j}^{N_x} w_{x,i} \times \sigma\Big(f(\phi(x, y'_i)) - f(\phi(x, y_i))\Big)\sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i))\Big)
\end{aligned}$$

We can use the symmetry in the expressions to simplify them somewhat.[5] Boosting step:

$$\begin{aligned}
\{\mathbf{Leaf}_r, b_r\}_{r=1}^R &= \operatorname*{arg\,min}_{h \in \mathcal{H}_{\text{tree}}(d)} \sum_{(x,y)} \sum_{j=1}^m l''_j(f^\phi(x), y) \left(-\frac{l'_j(f^\phi(x), y)}{l''_j(f^\phi(x), y)} - h(\phi(x, j))\right)^2 \\
\hat{\gamma}_r &= -\frac{\sum_{(x,y)} \sum_{j=1:\ \phi(x,j) \in \mathbf{Leaf}_r}^m l'_j(f^\phi(x), y)}{\sum_{(x,y)} \sum_{j=1:\ \phi(x,j) \in \mathbf{Leaf}_r}^m l''_j(f^\phi(x), y)} \qquad \forall r = 1 \dots R
\end{aligned}$$

B Proofs

Proof of Lemma A.1. We will relate the following three objectives for $h : \mathcal{X} \to \mathbb{R}^m$,

$$\begin{aligned}
J_1(h) &= \mathbf{E}\left[l(f(X) + h(X), Y)\right] = L(f + h) \\
J_2(h) &= \mathbf{E}\left[l(f(X), Y) + \nabla l(f(X), Y)^\top h(X) + \frac{1}{2} h(X)^\top \nabla^2 l(f(X), Y) h(X)\right] \\
J_3(h) &= \mathbf{E}\left[\left(-\nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) - h(X)\right)^\top \nabla^2 l(f(X), Y) \left(-\nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) - h(X)\right)\right]
\end{aligned}$$

$J_2(h)$ is an approximation of $J_1(h)$ obtained by plugging in the second-order Taylor approximation $l(f(x)+h(x), y) \approx l(f(x), y) + \nabla l(f(x), y)^\top h(x) + \frac{1}{2} h(x)^\top \nabla^2 l(f(x), y) h(x)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. $J_3(h)$ is equivalent to $J_2(h)$, more explicitly

$$\begin{aligned}
J_3(h) &= \mathbf{E}\left[\left(-\nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) - h(X)\right)^\top \nabla^2 l(f(X), Y) \left(-\nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) - h(X)\right)\right] \\
&= \mathbf{E}\left[\left(-\nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) - h(X)\right)^\top \left(-\nabla l(f(X), Y) - \nabla^2 l(f(X), Y) h(X)\right)\right] \\
&= \mathbf{E}\left[\nabla l(f(X), Y)^\top \nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y) + 2 \nabla l(f(X), Y)^\top h(X) + h(X)^\top \nabla^2 l(f(X), Y) h(X)\right] \\
&= \mathbf{E}\left[\nabla l(f(X), Y)^\top \nabla^2 l(f(X), Y)^{-1} \nabla l(f(X), Y)\right] + 2 J_2(h) - 2\mathbf{E}\left[l(f(X), Y)\right]
\end{aligned}$$

so that $\arg\min_{h \in \mathcal{H}} J_3(h) = \arg\min_{h \in \mathcal{H}} J_2(h)$. Assuming $\mathcal{H}$ is sufficiently expressive, the optimal regressor h^* for J_3 (and J_2) predicts for each $x \in \mathcal{X}$

$$h^*(x) = \operatorname*{arg\,min}_{z \in \mathbb{R}^m} \mathbf{E}\left[\left(-\nabla^2 l(f(x), Y)^{-1} \nabla l(f(x), Y) - z\right)^\top \nabla^2 l(f(x), Y) \left(-\nabla^2 l(f(x), Y)^{-1} \nabla l(f(x), Y) - z\right)\right]$$

[5] Define $\lambda_{x,i}(f) := -w_{x,i} \times \sigma(f(\phi(x, y'_i)) - f(\phi(x, y_i)))$, then we can write

$$\begin{aligned}
l'_j(f^\phi(x), y) &= \sum_{i=1:\ y_i=j}^{N_x} \lambda_{x,i}(f) - \sum_{i=1:\ y'_i=j}^{N_x} \lambda_{x,i}(f) \\
l''_j(f^\phi(x), y) &= -\sum_{i=1:\ y_i=j}^{N_x} \lambda_{x,i}(f) \sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i))\Big) - \sum_{i=1:\ y'_i=j}^{N_x} \lambda_{x,i}(f) \sigma\Big(f(\phi(x, y_i)) - f(\phi(x, y'_i))\Big)
\end{aligned}$$

Let $\pi : \mathbb{R}^m \to \mathbb{R}$ denote the objective for z here. The gradient and Hessian of $\pi(z)$ are

$$\begin{aligned}\nabla \pi(z) &= 2\mathbf{E}\left[\nabla l(f(x), Y)\right] + 2\mathbf{E}\left[\nabla^2 l(f(x), Y)\right] z \\ \nabla^2 \pi(z) &= 2\mathbf{E}\left[\nabla^2 l(f(x), Y)\right]\end{aligned}$$

which implies that π is convex (i.e., $\nabla^2 \pi(z) \succeq 0$) if l is convex (in the first argument). In this case, the optimal prediction is given by

$$h^*(x) = -\mathbf{E}\left[\nabla^2 l(f(x), Y)\right]^{-1} \mathbf{E}\left[\nabla l(f(x), Y)\right]$$

$\square$