Lecture 12:

Meta-Learning Methods

Sam Roweis

November 25, 2003

Meta-Learning

- The idea of meta-learning is to come up with some procedure for taking a learning algorithm and a fixed training set, and somehow repeatedly applying the algorithm to *different* subsets (weightings) of the training set or using *different* random choices within the algorithm in order to get a large ensemble of machines.
- The machines in the ensemble are then *combined* in some way to define the final output of the learning algorithm (e.g. classifier)
- The hope of meta-learning is that it can "supercharge" a mediocre learning algorithm into an excellent learning algorithm, without the need for any new ideas!
- There is, as always, good news and bad news....
  - The Bad News: there is (quite technically) No Free Lunch.
  - The Good News: for many real world datasets, meta learning works very well.

Meta-Learning Cafeteria

- Many meta-learning methods that work well in practice.
- We will review the three main ones:
  - Bagging: apply your algorithm to bootstrap datasets and average the predictions of the resulting ensemble.
  - Stacking: define a set of models by restricting the input to subsets of various sizes. Use LOO-CV to choose weights which blend these models.
  - Boosting: iteratively reweight your dataset, placing higher weights on the examples you are getting wrong. At each iteration, refit and add the result to your ensemble.

Why does Meta-Learning Work?

- Either reduces variance substantially without affecting bias (bagging, stacking), or vice versa (boosting).
- All meta-learning is based on one of two observations:
  A) Variance Reduction: *If we had completely independent training sets* it always helps to average together an ensemble of learners because this reduces variance without changing bias.
  B) Bias Reduction: For many simple models, a weighted average of models has much greater capacity than a single model (e.g. hyperplane classifiers, single-layer networks, Gaussian densities). So averaging models can often reduce bias substantially by increasing capacity.

Variance Reduction by Averaging

- Here is an example of how to show that averaging across independent training sets always reduces expected squared error:

$$
\begin{aligned}
e\bar{r}r_1 &= \sum_{x,y} p(x,y)\,(y - f(x|ts_1))^2 \\
e\bar{r}r &= \langle\langle \Big[y^2 - 2yf(x|ts) + f^2(x|ts)\Big]\rangle_{x,y}\rangle_{ts} = \langle e\bar{r}r_1\rangle_{ts} \\
f_{meta}(x_{test}) &= \frac{1}{T}\sum_i f(x_{test}|ts_i) = \langle f(x_{test}|ts)\rangle_{ts} \\
e\bar{r}r_{meta} &= \sum_{x,y} p(x,y)(y - \langle f(x|ts)\rangle_{ts})^2 \\
&= \langle \Big[y^2 - 2y\langle f(x|ts)\rangle_{ts} + (\langle f(x|ts)\rangle_{ts})^2\Big]\rangle_{x,y} \\
&\le \langle \Big[y^2 - 2y\langle f(x|ts)\rangle_{ts} + \langle f^2(x|ts)\rangle_{ts}\Big]\rangle_{x,y} \\
&\le e\bar{r}r \qquad\qquad \text{since } \langle f\rangle^2 \le \langle f^2\rangle
\end{aligned}
$$

Bagging (Breiman 1994)

- Bagging $\equiv$ **b**ootstrap **agg**regation.
- Idea is simple. Generate B bootstrap samples from your original training set. Train on each one to get f_b. Now average them:

$$f_{bag} = \frac{1}{B}\sum_b f_b$$

- For regression, average predictions, for classification, average class probabilities or take the majority vote if only hard outputs available.
- Bagging approximates the Bayesian posterior mean. The more bootstraps you use, the better, so use as many as you have time for.
- The size of each bootstrap sample is equal to the size of the original training set, but they are drawn *with replacement*, so each one contains some duplicates of certain training points and leaves out other training points completely.

Bagging Can Hurt

- Bagging helps when a learning algorithm is good on average but *unstable* with respect to the training set.
- But if we bag a stable learning algorithm, we can actually make it worse. For example, if we have a Bayes optimal algorithm, and we bag it, we might leave out some training samples in every bootstrap, and so the optimal algorithm will never be able to see them.
- Bagging almost always helps with regression, but even with unstable learners it can hurt in classification. If we bag a poor unstable classifier we can make it horrible.
- Example: true class = A for all inputs.
  Our learner guesses class A with probability 0.4 and class B with probability 0.6 regardless of the input. (Very unstable!).
  It has error 0.6.
  But if we bag it, it will have error 1.

Stacking (Wolpert 1990)

- In bagging, we created an ensemble of models by creating many synthetic training sets using the bootstrap.
- We can also create an ensemble of models in other ways, e.g. by restricting each model to look at only a subset of inputs, by trying the whole "kitchen sink" of regressors or classifiers (e.g. neural nets vs. logistic regression vs. naive bayes vs. KNN).
- In *stacked generalization* or *stacking* we try to find the best nonuniform weights to average our models together:

$$f_{stack}(x) = \sum_m w_m f_m(x)$$

- How should we set the weights? Using training error of each model? No! This will put too much weight on the most complex models.

Setting the Stacking Weights

- We estimate the optimal weights by setting them to minimize the average leave-one-out cross validation error:

$$w_m^* = \arg\min_w \sum_{i=1}^{N} \left[y_i - \sum_m w_m f_m^{-i}(x_i) \right]^2$$

  where f_m^{-i} is the result of model m trained on all points except i.
- These weights can be found exactly using linear regression.
- This is like a generalization of model selection using LOO-CV. Previously we picked the best model and set $w_{mbest} = 1$ and all other $w_m = 0$. Now we are doing a smooth weighting.
- In more advanced stacking ideas, we can combine the models nonlinearly and use weights which depend on the input x. This is like a mixture of experts where we fit the gate using cross-validated training points instead of the usual training set.

AdaBoost Algorithm

- Set initial observation weights $w_i = 1/N$.
- Loop while $(err_m < .5)$ $\{$
  - Fit the base classifier to the training data weighted by w_i. This results in the m^{th} round classifier $f_m(x)$.
  - Compute $err_m = \sum_i w_i e_{mi} / \sum_i w_i$ ($e_{mi} = 1$ if sign$[y_i] \neq$sign$[f_m(x_i)]$)
  - Set $\alpha_m = \frac{1}{2}\log[(1 - err_m)/err_m]$
  - Set $w_i \leftarrow w_i \exp[2\alpha_m e_{mi}]$
  - $m \leftarrow m + 1$

  $\}$
- Final classifier:

$$f_{boost}(x) = \text{sign}\left[\sum_m \alpha_m f_m(x)\right]$$

Boosting (Shapire 1990)

- Probably one of the three most influential ideas in machine learning in the last decade, along with Kernel methods and Variational approximations.
- In the PAC framework, boosting is a way of converting a "weak" learning model (behaves slightly better than chance) into a "strong" learning mode (behaves arbitrarily close to perfect).
- Very amazing theoretical result, but also lead to a very powerful and practical algorithm which is used all the time in real world machine learning.
- Basic idea, for binary classification with $y = \pm 1$.

$$f_{boost}(x) = \text{sign}\left[\sum_m \alpha_m f_m(x)\right]$$

  where $f_m(x)$ are models trained with reweighted datasets D_m, and the weights α_m are non-negative.

Forward Stagewise Additive Modeling

- Recall the additive model setup:

$$f_{add}(x) = \sum_m \alpha_m f_m(x; \theta_m)$$

- The overall function is a weighted sum of simpler functions, each with their own set of parameters.
  e.g.: hidden units in a MLP, wavelets, nodes in trees
- The optimization problem of finding the best $\{\alpha\}$ and $\{\theta\}$ simultaneously is usually extremely hard.
- But we can use a *greedy approximation*:
  - Initialize $f_0 = 0$.
  - for $m = 1 : M$ $\{$

    set $\alpha_m, \theta_m = \arg\min_{\alpha,\theta} \sum_{i=1}^{N} \text{cost}[y_i, f_{m-1}(x_i) + \alpha f(x_i; \theta)]$

    set $f_m(x) = f_{m-1}(x) + \alpha_m f(x; \theta_m)$

    $\}$

Some Intuitions about Boosting

- At each round, boosting reweights the examples it is doing poorly on more highly.
- The weight each intermediate classifier gets in the final ensemble depends on the error rate it achieved on its weighted training set at the time it was created.
- The reweighting over observations selected by boosting at each round is such that the existing ensemble would perform at chance.

Boosting as Forward Additive Modeling

- At each round of boosting we must minimize:

$$\begin{aligned} C &= \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \alpha_m f(x_i;\theta_m))] \\ &= \sum_{i=1}^{N} w_i^m \exp[-\alpha_m y_i f(x_i;\theta_m)] \end{aligned}$$

  with respect to α_m and θ_m, where $w_i^m = \exp(-y_i f_{m-1}(x_i))$.
- The optimal function and weight are given by:

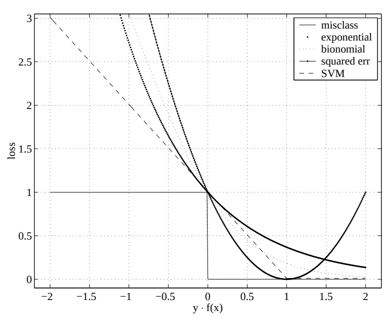
$$\begin{aligned} err_m &= \sum_{i=1}^{N} w_i^m [y_i \neq f(x_i;\theta_m)] / \sum_i w_i^m \\ \theta_m^*(x) &= \arg\min_\theta err_m \\ \alpha_m^* &= \frac{1}{2} \log \frac{1 - err_m}{err_m} \end{aligned}$$

Boosting tries to minimize Exponential Loss

- An amazing fact, which helps a lot to understand how boosting really works, is that classification boosting is equivalent to fitting a greedy forward additive model using the following cost function:

$$\text{cost}[y, f(x)] = \exp(-y f(x))$$

- This is called *exponential loss* and it is very similar to other kinds of loss, e.g. classification loss.



Updating the observation weights

- Finally, we update our approximation to get

$$f_m(x) = f_{m-1}(x) + \alpha_m^* f(x;\theta_m^*)$$

- This sets the new weights:

$$\begin{aligned} w_i^{m+1} &= w_i^m \exp[-\alpha_m y_i f(x_i;\theta_m^*)] \\ &= w_i^m \exp[\alpha_m(2e_{mi} - 1)] \\ &= w_i^m \exp[2\alpha_m e_{mi}] \exp[-\alpha_m] \end{aligned}$$

  where the last factor of $\exp[-\alpha_m]$ just rescales all the weights uniformly, so we can drop it.

More on Exponential Loss

- Exponential loss is very similar to other classification losses.
- It is minimized by setting $f(x)$ to one half the log-odds:

$$f^*(x) = \frac{1}{2}\frac{Prob[y=1|x]}{Prob[y=-1|x]}$$

  which means we can interpret $f(x)$ as the logit transform.
- Another loss function with the same population minimizer is the binomial negative log-likelihood:

$$-\log(1+\exp(-2yf(x)))$$

- But binomial loss places less emphasis on the bad cases (high negative margin), and so it is more robust when data is noisy.