

Combining Models

We've learned about a lot of basic models: least-squares regression, naive Bayes, perceptron, LR, SVM, decision trees, kNN, etc. Some of these work for both classification and regression, some are task specific. Some classifiers are easily extended to multiclass settings, some are not. Some can accept “weighted” inputs (different costs on classifications), some can not.

Today, we talk about:

1. Bagging: a method of improving the generalization ability of any (almost) learning algorithm.
2. Boosting: a method of turning a crummy learning algorithm into a good learning algorithm.
3. Reductions: a set of methods for using existing learning algorithms to solve bigger problems.

Bagging

Suppose we have an algorithm that likes to overfit. One solution would be to get lots of different data sets, train the model on all, and then vote. Bagging implements this idea by using the *bootstrap*.

Idea: generate 10 bootstrapped data sets, and train a model on each. Bootstrapping is just sampling with replacement.

The reason this works is because a bootstrapped sample “looks like” a different sample from the data generating distribution.

Pros: really easy to implement, linear time additional cost, no tunable hyperparameters.

Cons: only provides moderate improvement over well-regularized models, linear time prediction cost.

Also called an *ensemble*.

Boosting

Roots in computational learning theory. Suppose \mathcal{A} is a learning algorithm that sucks: only gets 51% accuracy no matter what data you send it. This is a “weak” learner. We turn it in to a “strong” learner by iteration. Train a model that gets 51% accuracy, then train a second model wherever the first model errors, etc. Overall error will always decrease.

Most famous algorithm: AdaBoost...

1. Initialize example weights D to $1/N$
2. Iterate T times:
 - (a) Train a weak learner h on weighted data D , achieving error ϵ_t
 - (b) Let $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

(c) Update weights:

$$\begin{aligned} D_n &= \frac{1}{Z} D_n \times \begin{cases} \exp[-\alpha_t] & h(x_n) = y_n \\ \exp[\alpha_t] & h(x_n) \neq y_n \end{cases} \\ &= \frac{1}{Z} D_n \exp[-\alpha_t y_n h(x_n)] \end{aligned}$$

3. Use averaged output

Can be shown that this actually “boost” the margin.

One disadvantage of boosting is that it is sometimes overly susceptible to noise.

Reductions

Want to turn an algorithm for solving a simple problem into an algorithm for solving a more complex problem.

Simple example: multiclass classification with a binary classifier. Several straightforward approaches (OVA/OVR and AVA/OVO/AP) and several non-straightforward approaches (ECOC, SECOC, etc.).

Costing: Suppose examples have weights. Then, repeat k times subsampling data with probability proportional to the weight (rejection sampling). And vote. Expected cost is proportional to error rates on individual classifiers.

Weighted-all-pairs: k classes, each comes with a cost. Standard multiclass is given by a vector of all ones except a zero at the correct class. Reduce to weighted binary. Let $L(v)$ be the number of classes with cost at most v . Let $d_i = \int_0^{c_i} dv 1/L(v)$. Train all pairs of binary classifiers with cost $|d_i - d_j|$ for classes i vs. j . Reduction rate: twice individual losses.