Ensembles and model combination

The Netflix competition mentioned earlier in the course was won by a large *ensemble* of different predictors. Ensembles of different types have also been used in many Kaggle competitions since. For example, the (unofficial) *Kaggle Ensembling Guide* by Kaggle user Triskelion, describes (in more detail than this note!) how to improve predictions on several Kaggle challenge datasets by using different ensembling ideas.

Combining the predictions of different models is used both to reduce overfitting and underfitting(!). We don't have time to do a thorough review of ensemble methods, but I wanted to point out these opposite motivations, and mention something about if and when you might consider ensembles.

Averaging predictions

Bayesian predictions are a form of model averaging, the predictions are averaged over all possible models, weighted by how plausible they are. In fact, some early papers referred to variational approximations to Bayesian predictions as *Ensemble Learning*[1]. Another way to approximate the integral for Bayesian predictions is with *Monte Carlo* methods. We sample S plausible settings of the parameters from the posterior distribution[2], and replace the average under the posterior with an empirical average under the samples:

$$\begin{aligned} p(y \,|\, \mathbf{x}, \mathcal{D}) &= \int p(y \,|\, \mathbf{x}, \mathbf{w})\, p(\mathbf{w} \,|\, \mathcal{D})\, \mathrm{d}\mathbf{w} \\ &= \frac{1}{S} \sum_{s=1}^{S} p(y \,|\, \mathbf{x}, \mathbf{w}^{(s)}), \qquad \mathbf{w}^{(s)} \sim p(\mathbf{w} \,|\, \mathcal{D}). \end{aligned}$$

We are now explicitly writing the prediction as an average under S different predictors. Each individual predictor extrapolates in plausible ways. Combining the distributions gives a broader distribution, correctly (assuming the model is good) reflecting the range of where the new output y might be.

Most prediction competitions will make us report an actual guess of the new output y, rather than specifying a distribution. How do we proceed? The competition will define a loss function, such as "mean square error", on which we will be judged. In a real application, part of your task as a datascientist is to understand what an appropriate loss function is: how much should outliers be punished? Given a loss function $L(\hat{y}, y)$, specifying how bad it is if we guess $\hat{y}$ when the actual output is y, we can then minimize our expected loss:

$$\mathbb{E}_{p(y \,|\, \mathbf{x}, \mathcal{D})}[L(\hat{y}, y)] = \int L(\hat{y}, y)\, p(y \,|\, \mathbf{x}, \mathcal{D})\, \mathrm{d}y.$$

To minimize square loss we would report the mean of the predictive distribution, which we could approximate by averaging the point predictions from different members of an ensemble. Different loss functions suggest different procedures: to minimize the expected absolute error, we would report the median of the predictive distribution.

Bagging

Bagging, or *Bootstrap Aggregation*, is another model averaging strategy. It's really easy to implement.

In the simplest version, the *Bootstrap* algorithm takes a training set with N points, and creates another training set of size N by sampling with replacement from the original datapoints.

---

1. For example, Ensemble learning in Bayesian neural networks, Barber and Bishop, 1998.
2. It's not obvious how to draw samples from the posterior distribution, unless the model is Gaussian, in which case we don't need to approximate the integral. We can use *Markov chain Monte Carlo* (MCMC) methods to draw posterior samples for models like logistic regression and neural networks. MCMC has been part of the MLPR course in previous years, but won't be on the exam this year. If you're interested, there is a tutorial you can watch online: `http://videolectures.net/mlss09uk_murray_mcmc/`

We'll usually select some training points more than once, and not include other datapoints. We fit a model to this perturbed dataset, and repeat the whole procedure S times, obtaining S different fitted models. At test time, we then average the S predictions from these models.

Bagging is meant for use with models that extrapolate wildly, such as high-order polynomials. The predictions at some input location will sometimes be far too big or far too small, depending on the details of the training set. That is, the fitting procedure has high variance (over different datasets). Averaging the fits from the bootstrapped datasets reduces the variance, giving more moderate predictions.

The different predictors in bagging result from the "best" fits to different plausible perturbations of the training data, rather than finding different plausible models of the one dataset we actually have. For scientific data analysis I personally prefer the Bayesian approach: I want to take a model seriously, and report beliefs about its parameters. Bagging is simpler however.

Both bagging and Bayesian model averaging assume that the model being fitted is sensible, but that there is not enough data to constrain it. If the model is too simple, and over-constrained by the data, then all of the posterior or bootstrap samples will be similar. The averaged predictions will be close to that of a single fit. If we'd like to build sophisticated models out of simpler pieces, Bayesian model averaging and bagging aren't ways to do it.

Combining models

A simple model may work well in a restricted region of the input space. For example, many functions are close to linear in a small region. If we have a set of models, such as simple linear regression models, we could use a neural network classifier to decide which model to use. Or we can use the probabilities from the classifier to weight the predictions of each predictor. This model is known as a "mixture of experts". It's a latent variable model, where each datapoint has an unknown class z, which is summed over to make predictions:

$$p(y \mid \mathbf{x}, \theta) = \sum_{k=1}^{K} p(y \mid \mathbf{x}, z=k, \theta)\, p(z=k \mid \mathbf{x}, \theta).$$

Here θ are parameters, and a neural network classifier $p(z \mid \mathbf{x}, \theta)$ weights the predictions of each "expert" in an ensemble. The whole system—the neural network and the regression models (or "experts")—can be trained jointly with gradient methods or EM. If worried about over-fitting, we could be Bayesian and integrate over the parameters, or we could use bagging.

Another way to divide up the input space is to use a *decision tree*, a series of simple rules, often based on individual features. For example, "if the person's age is greater than 40, and they are male, and they have a family history of cancer, then make decision X". Each statement (usually a binary choice) leads to a different path down the tree and a different decision. A regression tree gives a different regression model at each leaf of the tree. The tree as a whole will represent a piecewise-constant or piecewise-linear function. Decision trees are flexible: a long sequence of rules can specify small regions of the input space and so represent nearly arbitrary functions. As a result, decision trees can overfit, and bagging is useful. Bagged ensembles of trees with randomly-selected features are popular; you may have heard of their commercial variants, *Random Forests*™. There are also Bayesian treatments of decision trees.

Another family of algorithms for combining models is known as *boosting*. Components are added to a model to improve the fit of training points that are not predicted well by the current ensemble. Unlike bagging, boosting is usually used with simple models, which are combined to create a more complicated prediction function. When boosting algorithms are used with decision trees, the trees are usually restricted to small depth (or even a "decision stump" with only one rule).

A popular package called XGBoost has been used to win a large number of prediction competitions. It combines a boosting algorithm (gradient boosting) to build complex tree-based models with random sampling to avoid overfitting. It's fast, so people can try lots of variants easily. By using splits of real-valued, messy features, decision trees can learn functions that would be harder to fit with neural networks. Whereas neural networks tend to win competitions on well-understood domains like images, audio and text, where it's known how to process the data, and specialized neural network architectures have been developed.

What about interpretable and small models?

The large ensembles produced by competitors in prediction competitions are often criticized as being impractical, or hard to interpret.

One idea is to try to mimic a large ensemble that works well with a smaller and faster model. That is sometimes possible, e.g., *Model Compression*, Bucilă et al., KDD 2006. Fitting an ensemble with a decision tree might give interpretable rules that could be scrutinized for legal/ethical compliance.

In some applications, we might really want the most interpretable model of all: simple regression where the weights indicate whether a feature increases or decreases the output. Then we can check whether the fitted correlations match our causal understanding of the domain. The paper *Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission* (Caruana et al., SIGKDD 2015) gives a case study where interpretable transformations of features were learned, from which logistic regression could predict hospital patient outcomes (nearly) as well as neural networks.

Open Challenges?

I'll finish with some things we *can't* do (which is plenty).

New machine learning architectures (especially relating to neural networks) and regularization procedures are being proposed at a rate that is hard to keep up with. However, for a new task it's often not clear what ideas can be ignored, and what we should try, and human expertise is usually required to make things work.

Within closed domains, some hyper-parameter searches and choices are made using Bayesian optimization. But the big decisions about what experiments to run and how to spend computer time are still largely guided by humans. There is no one systematic procedure for how to solve a prediction problem, or win a Kaggle competition (if there were, the competitions wouldn't exist!). Although there are ambitious research programmes such as Ghahramani's "Automated Statistician", which aim to automate more parts of the datascience process than we have historically.

It's also not clear what computational expense is really required. If we can fit a small model to mimic a large ensemble, is there a more direct way of fitting the (regularized) small model without creating the enormous ensemble in between? Also, companies routinely refit large neural networks from scratch on new click and speech data. It should be possible to adapt methods online, as humans do, but apparently online learning methods in the literature are not robust enough.

Machine learning's currently an exciting area because of the amazing and useful things that can be done with today's methods. But on the research front there are plenty of open problems that will help enable tomorrow's technology. I hope some of you will be the drivers behind some of that technology.

Further Reading

Murphy Chapter 16 provides an overview of some flexible models, touching on both bagging and boosting.

The Hastie et al., *Elements of Statistical Learning* book (2nd Ed.) has a useful alternative treatment. Chapter 8 discusses bootstrap sampling and the relationship to Bayesian inference, with bagging in Section 8.7. Chapter 10 covers boosting.