Machine Learning, Spring 2019

Ensemble methods

Reading Assignment: Chapter 6 & 7

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

Ensemble methods

In previous lecture, we focused on interpretable classification with decision trees. But for many complex problems, a simple, interpretable classifier might not be optimal in terms of classification accuracy.

Many high-performing **black-box** classifiers exist (for example, deep learning) but these lack interpretability and may require both lots of processing power and lots of training data to achieve high performance.

Here's a simple way to get high performance, which also allows you to manage the accuracy vs. interpretability tradeoff: learn multiple, different predictors and let them **vote** (or **average** their outputs for regression).

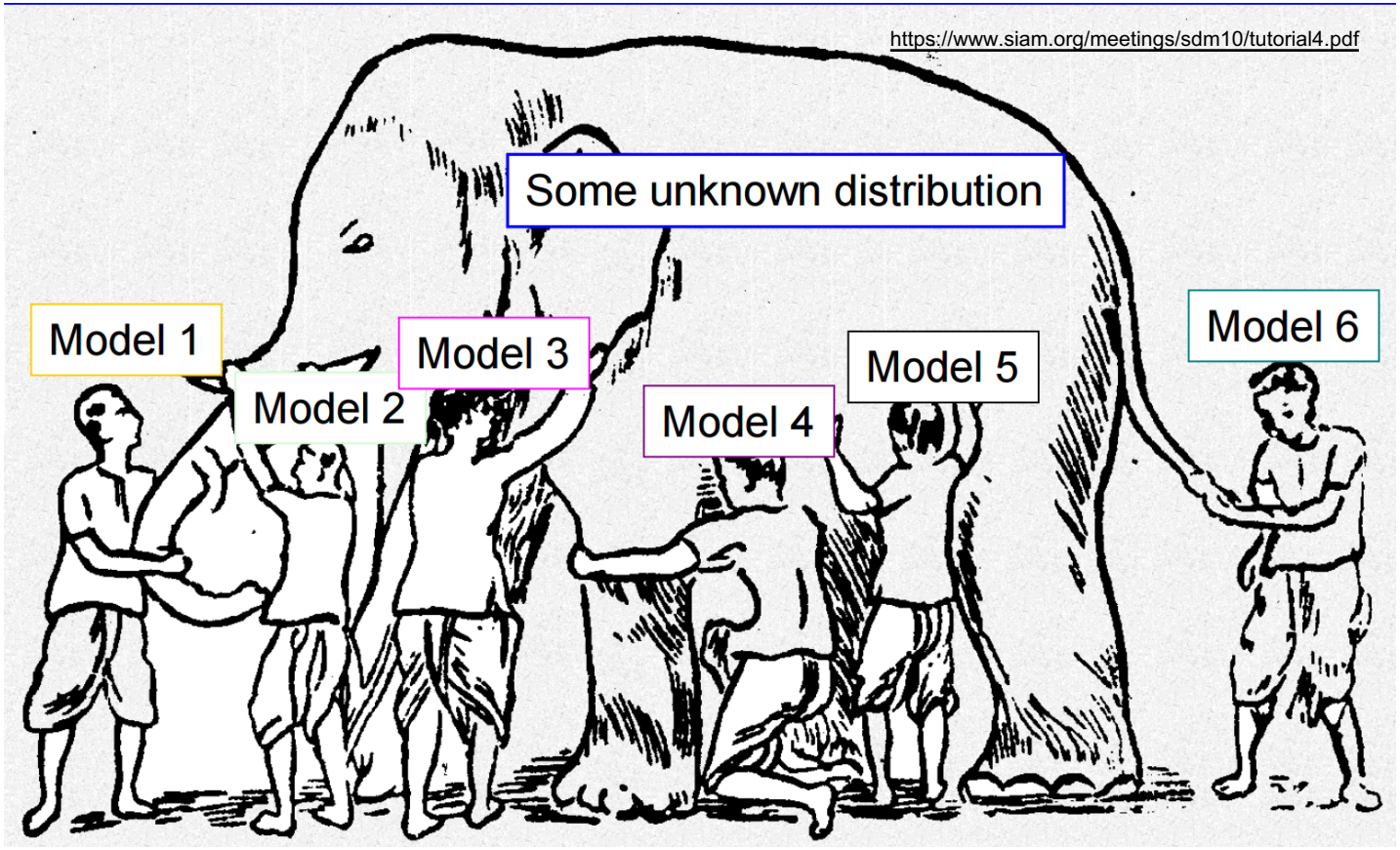
[Digression: majority voting or soft voting with probabilistic classifiers?]

Two natural ways to create multiple, different predictors from training data:

1) Learn different classes of models using the same training dataset.
2) Learn the same type of model (e.g., a decision tree) using different, randomly selected subsets of the training data.

Why might ensembles improve accuracy?

Think about making a decision by asking a panel of independent experts and taking a vote. Each expert has different knowledge of the data and (maybe) a different decision-making procedure. We might expect the majority decision to be better more often than asking any single expert.



Why might ensembles improve accuracy?

Ensembles are most useful when the predictors are maximally **independent**.

Example for regression: suppose h(x) is the true function we are trying to predict, and we have M separate models with errors.

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x})$$

The mean squared error of an individual model is:

$$\mathbb{E}_{\mathbf{x}}\left[\{y_m(\mathbf{x}) - h(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

Average error of all M models:

$$E_{\mathrm{AV}} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

Why might ensembles improve accuracy?

Ensembles are most useful when the predictors are maximally **independent**.

The ensemble's combined prediction is the unweighted average of the individual predictions. The expected value of the mean squared error is:

$$E_{\text{COM}} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right]$$
$$= \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

If the errors are **unbiased** and **uncorrelated**, then the MSE of the ensemble is smaller than the average MSE of the individual models by a factor of (1/M).

$$\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] = 0$$
$$\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_l(\mathbf{x})\right] = 0, \qquad m \neq l$$

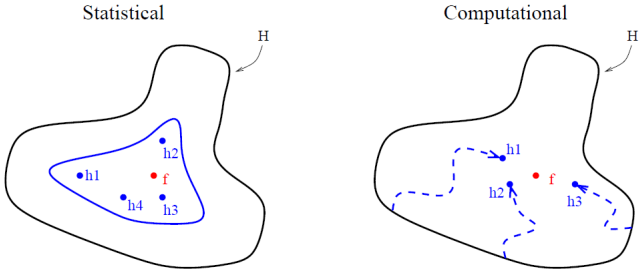
$\Rightarrow$

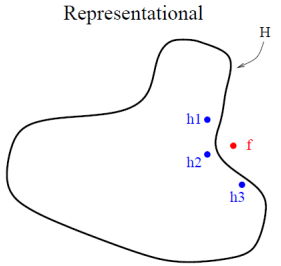
$$E_{\text{COM}} = \frac{1}{M} E_{\text{AV}}$$
$$= \frac{1}{M^2}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

What happens in the worst case scenario of perfectly correlated predictors?

Why might ensembles improve accuracy?

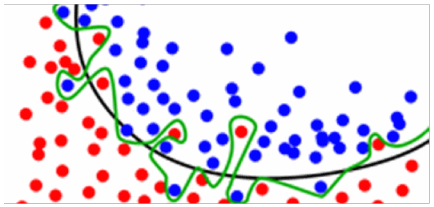
… more intuitions





Consider searching over a space of possible **hypotheses** H to identify the true function f mapping inputs x to outputs y. (Dietterich, 2000; Zhou, 2009)

Case 1 ("Statistical"): Insufficient training data to distinguish between multiple hypotheses h ∈ H. Each might have its own biases for generalization to unseen data, so best to average over these.



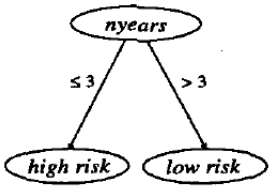
Case 2 ("Computational"): Some learners may converge on suboptimal hypotheses in H (e.g., by getting stuck in local optima, or drawing an unlucky sample of training data). Averaging helps to downweight these poor performers.

Case 3 ("Representational"): The hypothesis space being searched might not contain the true target function, while ensembles can create a good approximation.

Why might ensembles improve accuracy?

… from trees to forests

nyears	sports car	risk
1	*yes*	*high*
2	*no*	*high*
2	*yes*	*high*
4	*no*	*low*
8	*no*	*low*



(a) original sample and the corresponding tree

nyears	sports car	risk
1	*yes*	*high*
4	*yes*	*high*
2	*yes*	*high*
4	*no*	*low*
8	*no*	*low*

(b) changed sample and the corresponding tree

Figure 1: Different rules constructed from samples differing in one record

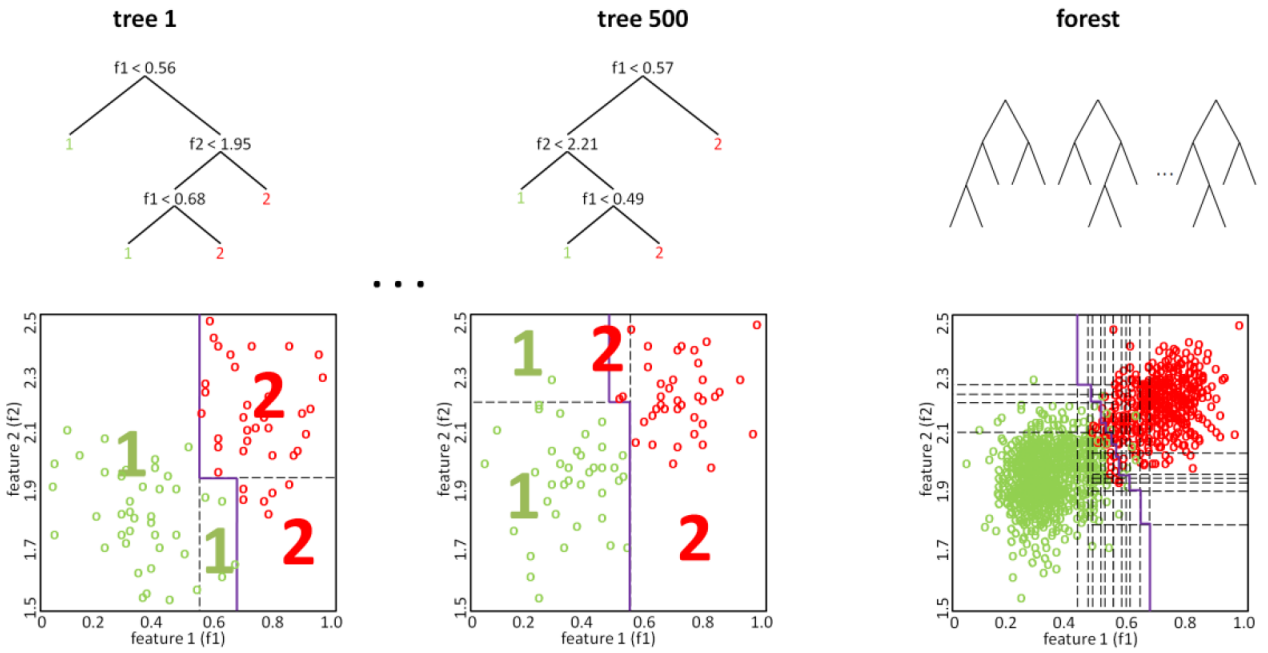
From Li and Belford, "Instability of decision tree classification algorithms", KDD 2002.

One disadvantage of decision tree classifiers is *instability*: small changes in the training dataset may lead to dramatic differences in the resulting tree, and therefore, in classification of previously unseen test examples.

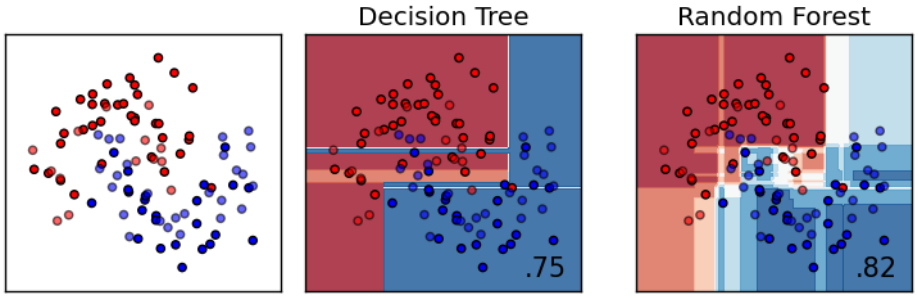
Averaging over multiple trees constructed from samples of the training data can mitigate this problem, improving both accuracy and stability.

Why might ensembles improve accuracy?

… from trees to forests



From Hanselmann et al., 2009.



From Martin Thoma (martin-thoma.com)

Averaging over multiple trees also allows better *representation* of non-axis-aligned splits.

Why might ensembles improve accuracy?

… in practice

Netflix competition: given a dataset of users and their movie ratings, predict a user's rating of any movie. $1M prize offered for winning team (have to beat Netflix's own algorithm by 10% RMSE).

→ The highest-performing teams all used ensemble methods!

Use of **random forests** is now ubiquitous: high performing, easy to implement, doesn't require careful parameter tuning or huge amounts of training data…

Random forests in the criminal justice system:

- Analyzing recidivism in Philadelphia's Adult Probation and Parole Department (Berk et al., 2009).
- Forecasting domestic violence (Berk et. al, 2016).
- Better pretrial detention policies (Shroff et al., in preparation).

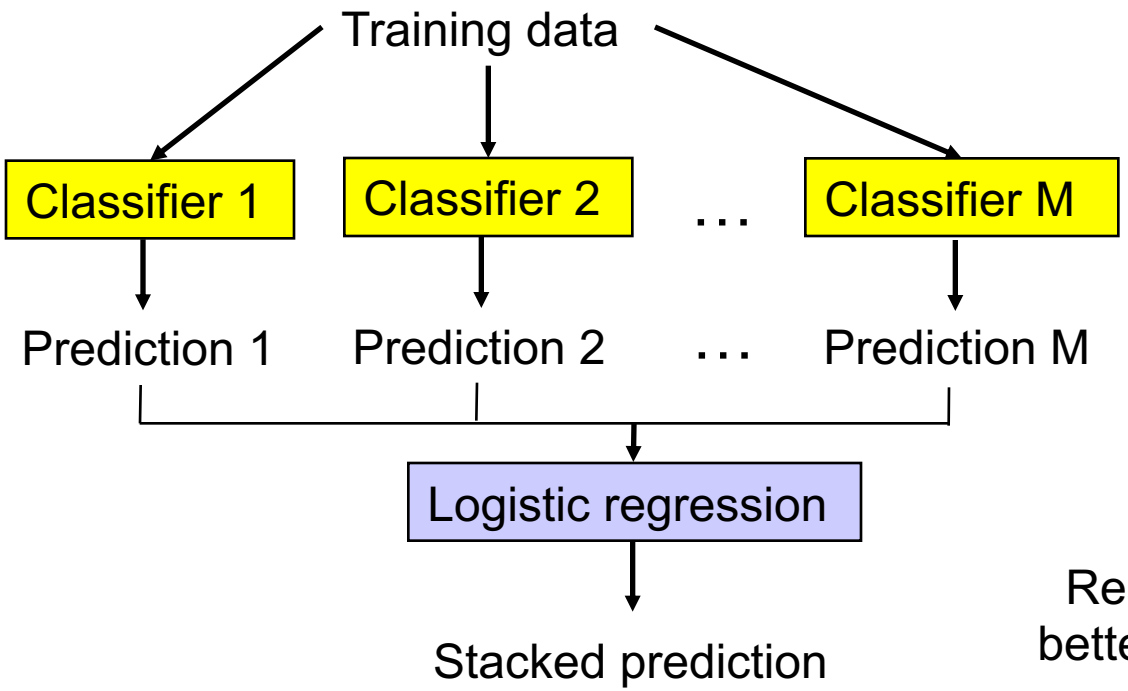
Q: What are some of the potential **disadvantages**?

A: lack of interpretability, increased computation, potential for overfitting.

Ensemble Methods 1: Stacking

Probably the simplest ensemble approach: learn a bunch of different models using the same training dataset, and let them vote (or average their predictions).

But an unweighted vote/avg typically underperforms the best individual model…



Solution: learn another classifier, typically logistic regression, to choose the weights of the individual classifiers.

Important to do this using a separate, held-out validation set (or cross-validation).

Resulting classifier is often slightly better than the best individual model: great for Kaggle competitions, not necessarily worth the effort in practice!

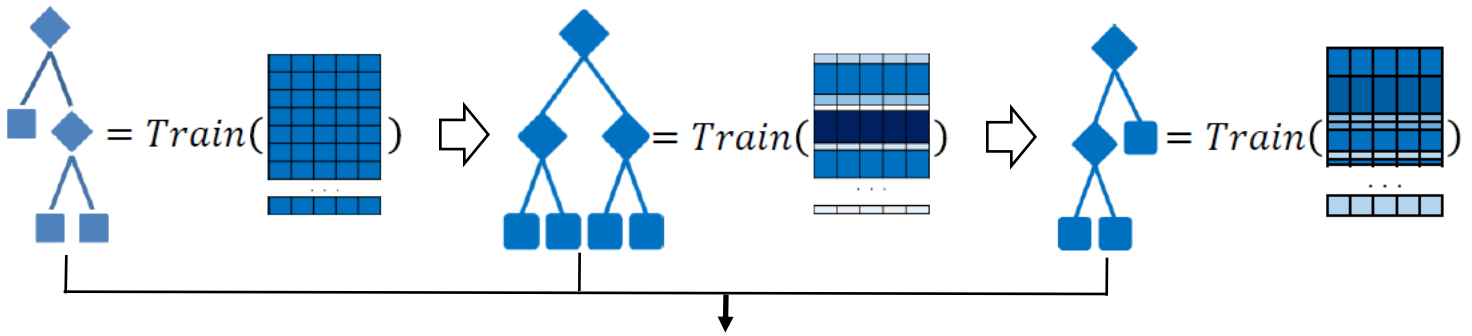
Ensemble Methods 2: Boosting

More complicated, but effective, ensemble methods where we learn a **sequence** of classifiers, each focusing on examples where previous models had difficulty.

Most common approach: **Adaboost**.
On each step, iteratively **reweight** the training data using the current set of models, giving exponentially higher weight to incorrectly predicted data points and lower weight to correctly predicted points, then learn a new model using the reweighted data. Final prediction = weighted avg of individual predictions.

Related and more general approach: **gradient boosting**.
Additive model: on each step, fit the model to the **residuals** left by fitting all previous models. This is equivalent to gradient descent in function space.

From Bilenko et al., KDD 2011 tutorial.



(add with weights proportional to log-odds of correct prediction)

Ensemble Methods 2: Boosting

More complicated, but effective, ensemble methods where we learn a **sequence** of classifiers, each focusing on examples where previous models had difficulty.

Most common approach: **Adaboost**.
On each step, iteratively **reweight** the training data using the current set of models, giving exponentially higher weight to incorrectly predicted data points and lower weight to correctly predicted points, then learn a new model using the reweighted data. Final prediction = weighted avg of individual predictions.

Related and more general approach: **gradient boosting**.
Additive model: on each step, fit the model to the **residuals** left by fitting all previous models. This is equivalent to gradient descent in function space.

Advantages: can start with **weak classifiers** (e.g., decision stumps) and get resulting strong classifier; reduces bias not just variance; good theoretical properties (minimize a convex loss function).

Disadvantages (as compared to random forests): harder to implement, less robust to outliers, sensitive to parameter values, and not easily parallelizable.

High accuracy: considered (one of) the best "out of the box" classifiers.

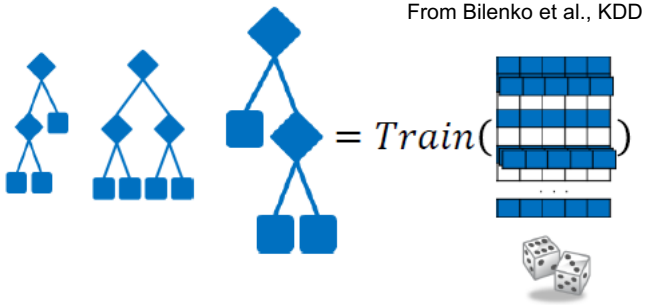
Ensemble Methods 3: Bagging

Short for “bootstrap aggregation”. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

Final prediction is an unweighted average (or vote) of the individual predictors.

Bootstrap sample: sample data records (rows) uniformly at random with replacement.

From Bilenko et al., KDD 2011 tutorial.



Advantages: much higher performance than individual trees, though often boosting or random forests will perform slightly better. Increases stability and reduces overfitting. Trivial to implement and to parallelize.

From Martinez-Muñoz and Suarez, 2010: while it is typical to use bootstrap samples of the same size as the original dataset, smaller bootstrap samples (e.g., 20-40% of original size) are sometimes better (and sometimes worse). Choose based on minimizing OOB on separate validation set.

(OOB = out-of-bag error = prediction error on each training sample x_i, using only trees not containing x_i.)

Ensemble Methods 4: Random Forests

A super-useful variant of bagging. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

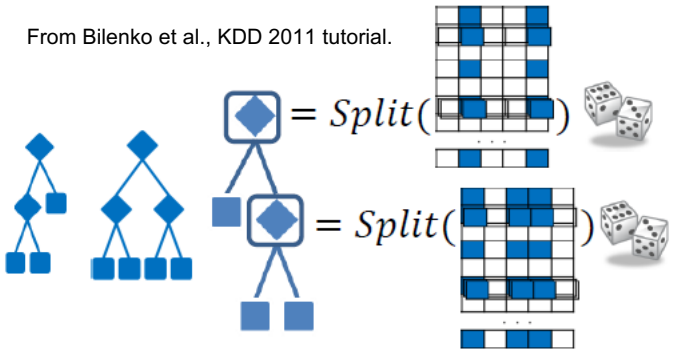
Final prediction is an unweighted average (or vote) of the individual predictors.

Key difference of RF from bagging:

When building each individual tree, each time we split, we restrict our choice to a randomly chosen subset of features (columns).

Typical choice: if original dataset has p dimensions, restrict to sqrt(p).

From Bilenko et al., KDD 2011 tutorial.



Advantages:

- Generally very accurate--see Delgado paper in references-and easy to use out-of-box.
- Efficiently parallelizable.
- Not prone to overfitting; no need to prune (low variance- trees aren't very correlated).
- With enough trees, can estimate OOB error.
- Can estimate feature importance.

Disadvantages:

- Computationally expensive. To train a model with N trees, m features, and n data points, complexity is O(Nm*n log(n)).
- Lots of memory to store trees.
- Not very interpretable.

Ensemble Methods 4: Random Forests

A super-useful variant of bagging. We learn a large set of models, e.g., decision trees, each using a different **bootstrap sample** from the training dataset.

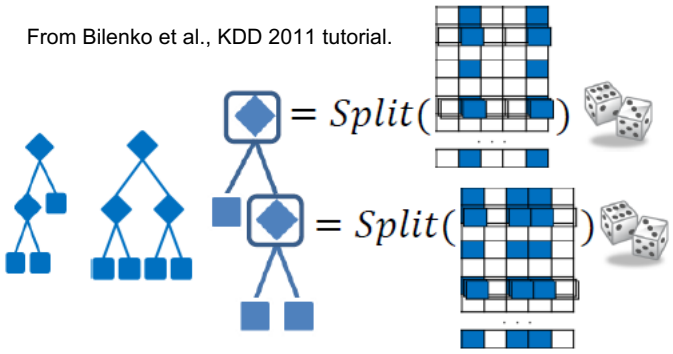
Final prediction is an unweighted average (or vote) of the individual predictors.

Key difference of RF from bagging:

When building each individual tree, each time we split, we restrict our choice to a randomly chosen subset of features (columns).

Typical choice: if original dataset has p dimensions, restrict to sqrt(p).

From Bilenko et al., KDD 2011 tutorial.



Choice of parameters:

- Number of trees in the forest
- Whether and how to prune the trees (min # of samples per leaf, max depth, min # samples to split, etc.)
- # of records and features to sample

But fairly robust to these choices!

Alternative approach (random subspaces)

Choose a single subset of features per decision tree rather than per split.

Empirically, random forests tend to do a bit better, but random subspaces are even more parallelizable– very useful for massive data-- and can be applied to prediction approaches other than trees.

Accuracy vs. interpretability in RF

When is interpretability important and when do we just need good accuracy?

More and larger trees = harder to interpret the model as a whole (e.g., which attributes are most important) and harder to interpret individual predictions.

For an individual prediction, you could imagine exhibiting individual trees that predict the majority class, or for regression, those giving individual predictions close to the ensemble's prediction. But which trees are most representative?

To interpret the model as a whole, one can post-process the ensemble in various ways to calculate measures of **variable importance**.

Gini importance: mean decrease in node impurity across all trees for splits on that variable (Breiman, 2001).

$$Imp(X_m) = \frac{1}{N_T} \sum_{T} \sum_{t \in T : v(s_t) = X_m} p(t) \Delta i(s_t, t)$$

Mean over trees

Sum over splits on that var.

Proportion of data pts reaching that split

Decrease in impurity, e.g., info. gain

Useful, but biased toward continuous vars or discrete vars with many values (Strobl et al., 2007)

Accuracy vs. interpretability in RF

When is interpretability important and when do we just need good accuracy?

More and larger trees = harder to interpret the model as a whole (e.g., which attributes are most important) and harder to interpret individual predictions.

For an individual prediction, you could imagine exhibiting individual trees that predict the majority class, or for regression, those giving individual predictions close to the ensemble's prediction. But which trees are most representative?

To interpret the model as a whole, one can post-process the ensemble in various ways to calculate measures of **variable importance**.

Permutation importance: mean decrease in overall accuracy (as measured on out-of-bag samples) when that variable is randomly permuted (Breiman, 2001).

Strobl et al. (2007) argue that permutation importance is biased toward correlated vars and describe a conditional permutation-based alternative (very computationally expensive).

Gregorutti et al. (2013) argue for utility of permutation importance in a backward elimination-based procedure for variable selection.

Accuracy vs. interpretability in RF

When is interpretability important and when do we just need good accuracy?

More and larger trees = harder to interpret the model as a whole (e.g., which attributes are most important) and harder to interpret individual predictions.

For an individual prediction, you could imagine exhibiting individual trees that predict the majority class, or for regression, those giving individual predictions close to the ensemble's prediction. But which trees are most representative?

To interpret the model as a whole, one can post-process the ensemble in various ways to calculate measures of **variable importance**.

Interesting alternative: learn a single decision tree that best predicts the output of the forest. Preserves interpretability along with a substantial fraction of the gains in accuracy and stability. See Domingos, 1998, for more details.

References

- Scitkit-learn documentation for random forests and other ensemble methods: http://scikit-learn.org/stable/modules/ensemble.html
- Ch. 14 of Bishop (ensemble methods).
- Ch. 10 of Hastie, Tibshirani, and Friedman (mainly about boosting).
- L. Breiman. Random forests. *Machine Learning*, 2001.
- M. Fernández-Delgado et al. Do we need hundreds of classifiers to solve real-world classification problems? *J. Mach Learn Res*., 2014.
- G. Martinez-Munoz and A. Suarez. Out-of-bag estimation of the optimal sample size in bagging. *Pattern Recognition*, 2010.
- M. Bilenko et al. Scaling up decision tree ensembles. KDD 2011 tutorial, http://hunch.net/~large_scale_survey/.
- C. Strobl et al. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 2007.
- B. Gregorutti et al. Correlation and variable importance in random forests. *Statistics in Computing*, 2016.
- P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 1998.

Up next: a short break, and then Python examples for trees and random forests.