Machine Learning, Spring 2020

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/

Acknowledge: The slides are partially referred to coursera online machine learning course by Prof. Andrew Ng, Prof. Zhouwen Tu (UCSD) and NYU machine learning course.

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).

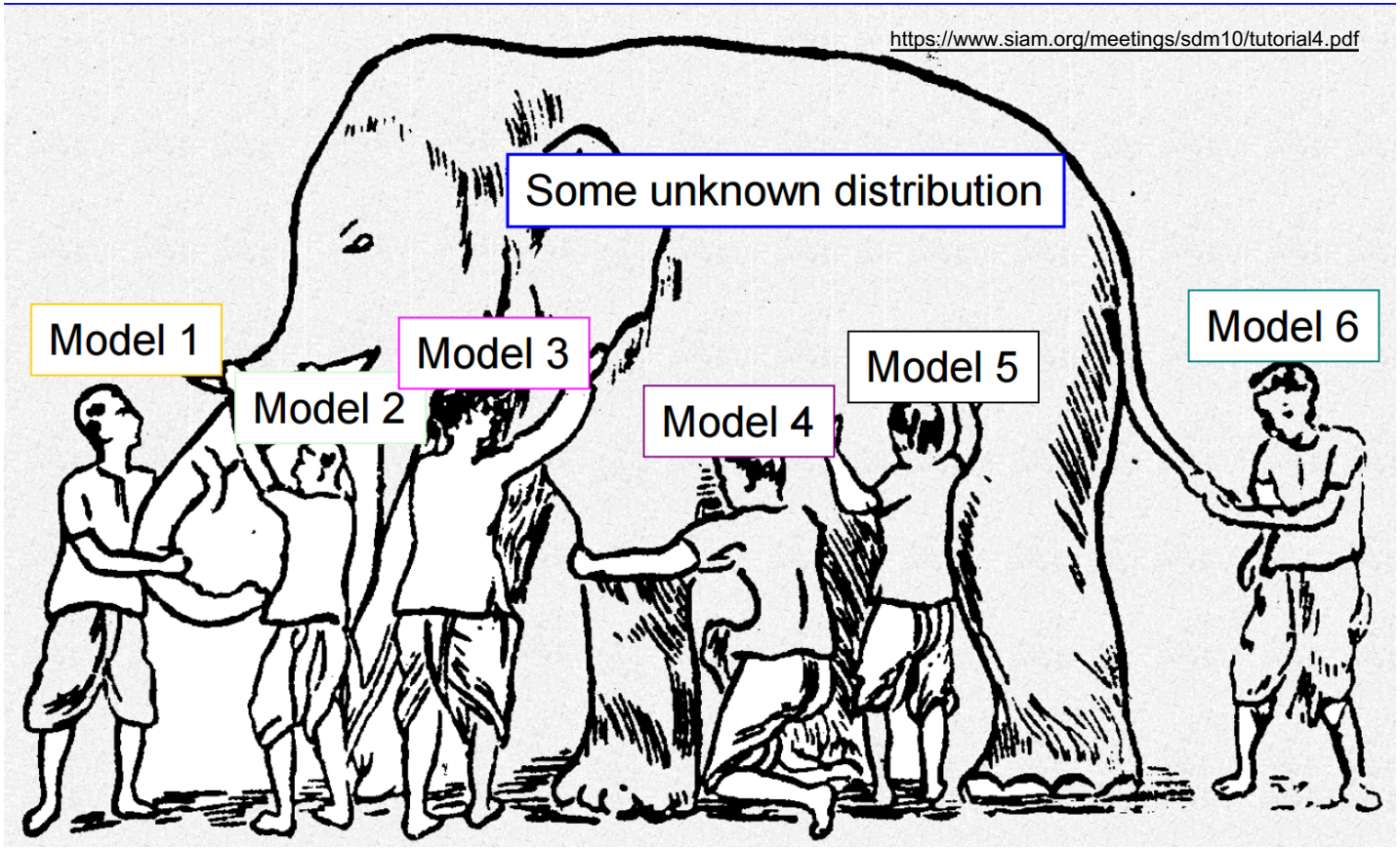
[Digression: majority voting or soft voting with probabilistic classifiers?]

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

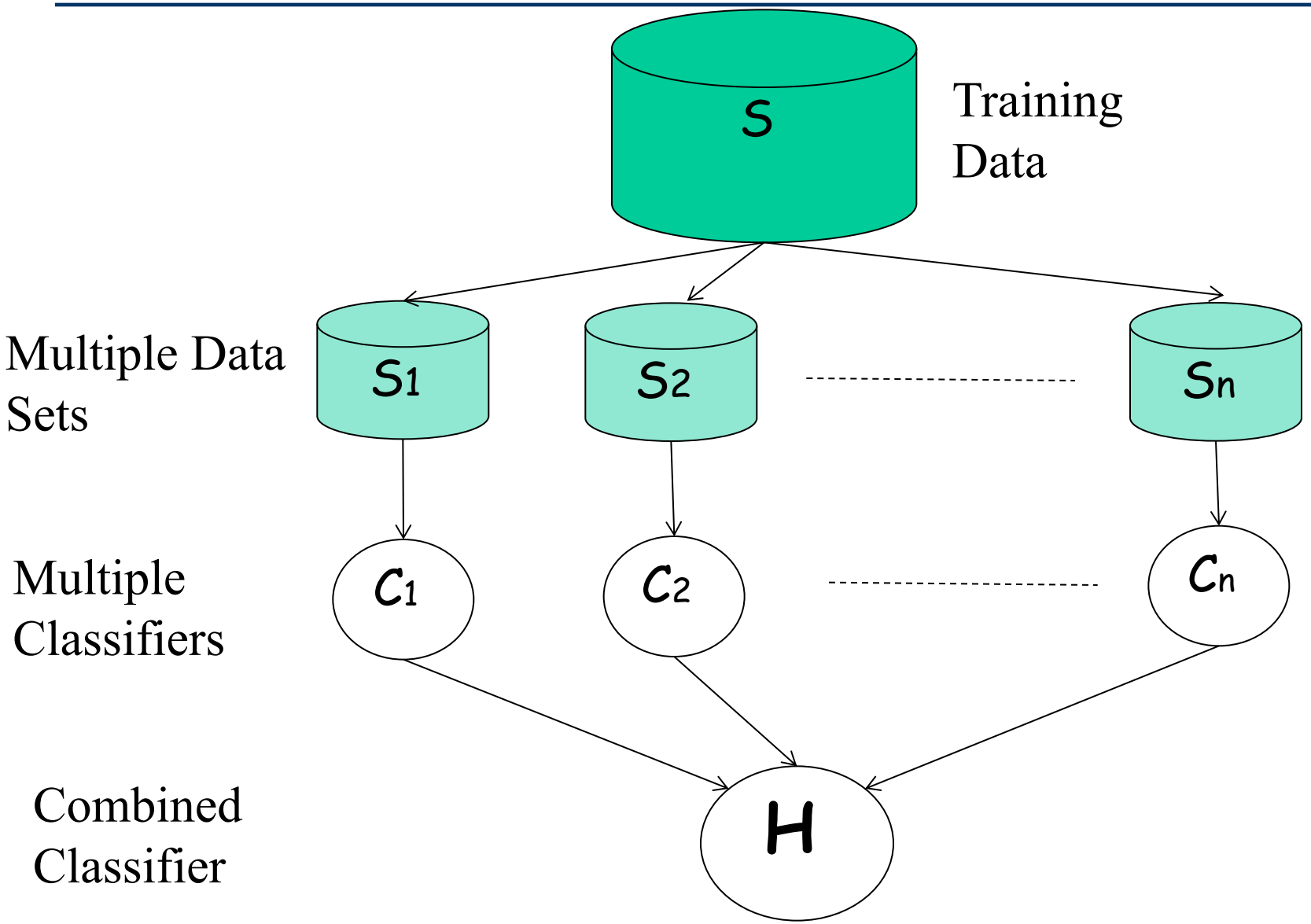
1) Learn different classes of models using the same training dataset.
2) Learn the same type of model (e.g., a decision tree) using different, randomly selected subsets of the training data.

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.



General Idea



Build Ensemble Classifiers

- Basic idea:

  Build different “experts”, and let them vote

- Advantages:

  Improve predictive performance

  Other types of classifiers can be directly included

  Easy to implement

  No too much parameter tuning

- Disadvantages:

  The combined classifier is not so transparent (black box)

  Not a compact representation

Why do they work?

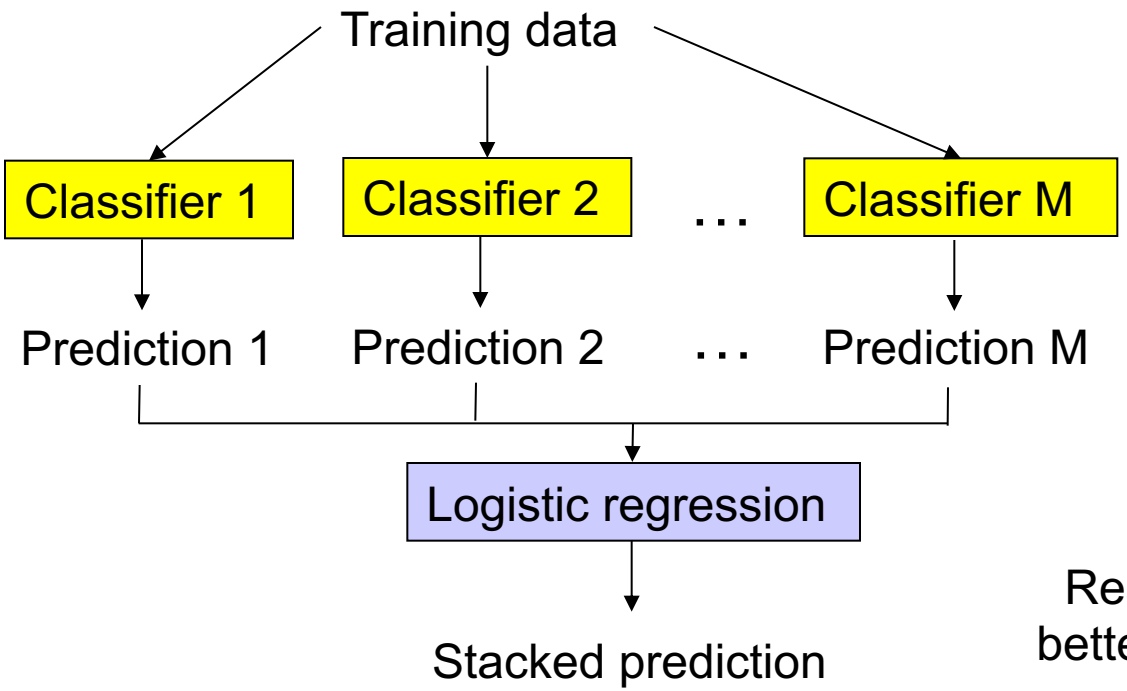
- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{\mathrm{i}} (1-\varepsilon)^{25-i} = 0.06$$

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!

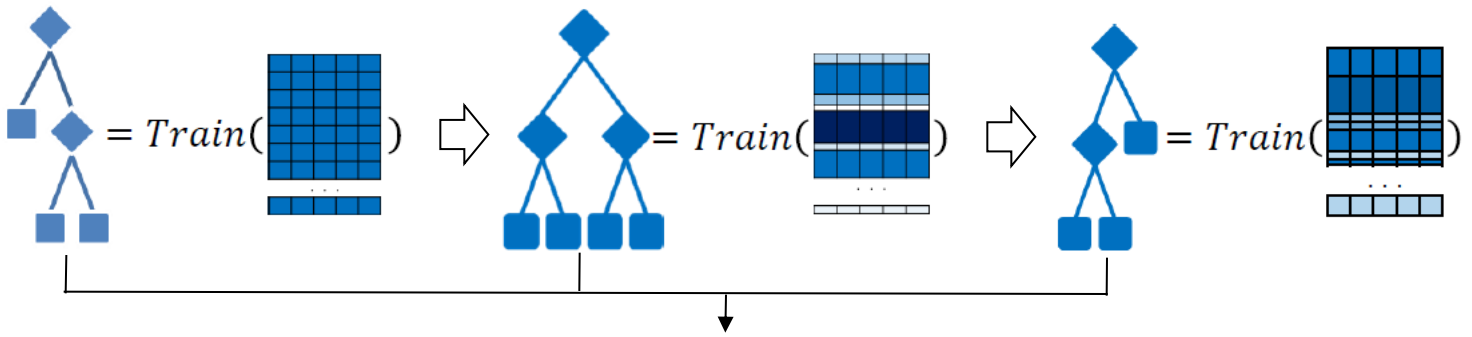
Ensemble Methods 2: Boosting

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.

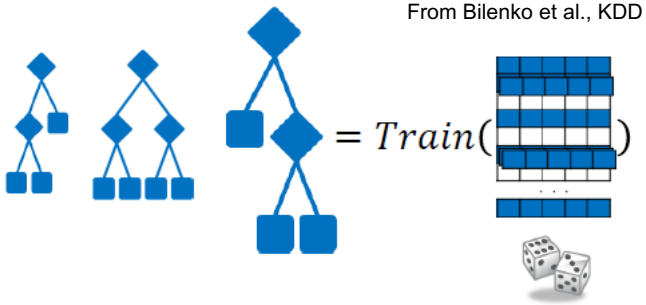
Ensemble Methods 3: Bagging

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.

From Bilenko et al., KDD 2011 tutorial.

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



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.

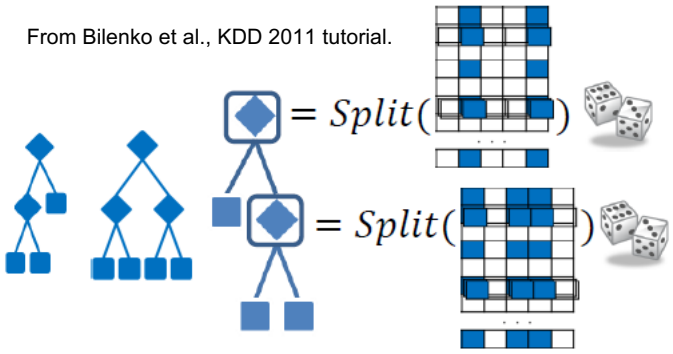
Final prediction is an unweighted average (or vote) of the individual predictors.

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.

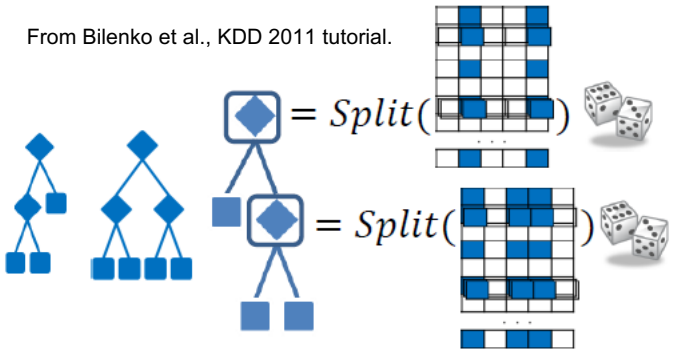
Final prediction is an unweighted average (or vote) of the individual predictors.

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.

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.