Combining multiple models

- Basic idea of “meta” learning schemes: build different “experts” and let them vote
- Advantage: often improves predictive performance
- Disadvantage: produces output that is very hard to analyze
- Schemes we will discuss: *bagging*, *boosting*, *stacking*, and *error-correcting output codes*
  - The first three can be applied to both classification and numeric prediction problems

Bagging

- Employs simplest way of combining predictions: voting/averaging
- Each model receives equal weight
- “Idealized” version of bagging:
  - Sample several training sets of size *n* (instead of just having one training set of size *n*)
  - Build a classifier for each training set
  - Combine the classifier’s predictions
- This improves performance in almost all cases if learning scheme is *unstable* (i.e. decision trees)

Bias-variance decomposition

- Theoretical tool for analyzing how much *specific* training set affects performance of classifier
- Assume we have an infinite number of classifiers built from different training sets of size *n*
  - The *bias* of a learning scheme is the expected error of the combined classifier on new data
  - The *variance* of a learning scheme is the expected error due to the particular training set used
  - Total expected error: bias + variance

More on bagging

- Bagging reduces variance by voting/averaging, thus reducing the overall expected error
  - In the case of classification there are pathological situations where the overall error might increase
  - Usually, the more classifiers the better
- Problem: we only have one dataset!
- Solution: generate new datasets of size *n* by sampling with replacement from original dataset
- Can help a lot if data is noisy

Bagging classifiers

model generation

```
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training set.
  Apply the learning algorithm to the sample.
  Store the resulting model.
```

classification

```
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
```

Boosting

- Also uses voting/averaging but models are weighted according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm

AdaBoost.M1

model generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Apply learning algorithm to weighted dataset and store
     resulting model.
  Compute error e of model on weighted dataset and store error.
  If e equal to zero, or e greater or equal to 0.5:
    Terminate model generation.
  For each instance in dataset:
    If instance classified correctly by model:
       Multiply weight of instance by e / (1 - e).
  Normalize weight of all instances.
```

classification

```
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add -log(e / (1 - e)) to weight of class predicted by model.
Return class with highest weight.
```

More on boosting

- Can be applied without weights using resampling with probability determined by weights
  - Disadvantage: not all instances are used
  - Advantage: resampling can be repeated if error exceeds 0.5
- Stems from *computational learning theory*
- Theoretical result: training error decreases exponentially
- Also: works if base classifiers not too complex and their error doesn't become too large too quickly

A bit more on boosting

- Puzzling fact: generalization error can decrease long after training error has reached zero
  - Seems to contradict Occam's Razor!
  - However, problem disappears if *margin* (confidence) is considered instead of error
    - Margin: difference between estimated probability for true class and most likely other class (between –1, 1)
- Boosting works with *weak* learners: only condition is that error doesn't exceed 0.5
- LogitBoost: more sophisticated boosting scheme

Stacking

- Hard to analyze theoretically: “black magic”
- Uses *meta learner* instead of voting to combine predictions of base learners
  - Predictions of base learners (*level-0 models*) are used as input for meta learner (*level-1 model)*
- Base learners usually different learning schemes
- Predictions on training data can’t be used to generate data for level-1 model!
  - Cross-validation-like scheme is employed

More on stacking

- If base learners can output probabilities it's better to use those as input to meta learner
- Which algorithm to use to generate meta learner?
  - In principle, any learning scheme can be applied
  - David Wolpert: "relatively global, smooth" model
    - Base learners do most of the work
    - Reduces risk of overfitting
- Stacking can also be applied to numeric prediction (and density estimation)