Model Ensembles

- **Basic idea:**
  Instead of learning one model, learn several and combine them
- Typically improves accuracy, often by a lot
- **Many methods:**
  - Bagging
  - Boosting
  - ECOC (error-correcting output coding)
  - Stacking
  - Etc.

---

**Bagging**

- Generate “bootstrap” replicates of training set by sampling with replacement
- Learn one model on each replicate
- Combine by uniform voting

---

**Boosting**

- Maintain vector of weights for examples
- Initialize with uniform weights
- Loop:
  - Apply learner to weighted examples (or sample)
  - Increase weights of misclassified examples
- Combine models by weighted voting

---

**AdaBoost**($S, \text{Learn}, k$)

* $S$: Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $y \in Y$
* **Learn:** Learner($S$, weights)
* $k$: # Rounds
For all $i$ in $S$: $w_0(i) = 1/m$
For $r = 1$ to $k$
do
  For all $i$: $w_r(i) = w_r(i) / \sum_i w_r(i)$
  $h_r = \text{Learn}(S, p_r)$
  $\varepsilon_r = \sum_i p_r(i) \mathbf{1}[h_r(i) \neq y_i]$
  If $\varepsilon_r > 1/2$ then
  $k = r - 1$
  Exit
  $\beta_r = \varepsilon_r / (1 - \varepsilon_r)$
For all $i$: $w_{r+1}(i) = w_r(i) \beta_r \mathbf{1}[h_r(i) \neq y_i]$
Output: $h(x) = \arg\max_{y \in Y} \sum_{r=1}^{k} (\log \frac{1}{\beta_r}) \mathbf{1}[h_r(x) = y]$
How Will # of Rounds Effect Generalization?

Expect
- Training error to drop or reach 0
- Test error to increase when h* becomes too complex: "Occam's razor" (i.e., overfitting)
- Hard to know when to stop training

Empirical Results
- Often, test error does not increase, even after 1000 rounds!
- Test error continues to drop, even after training error is 0!
- Occam’s razor: "simpler is better" appears to not apply!
**Explanation: Margins**

- **Key idea:**
  - Training error only measures whether classifications are right or wrong
  - Should also consider confidence of classifications
- **h** is weighted majority vote of weak classifiers
- **Measure confidence by margin:** Strength of vote
  - \((\text{weighted vote } +) - (\text{weighted vote } -)\)

**AdaBoost Advantages**

- Fast, simple and easy to program
- No parameters to tune (except T, sometimes)
- Flexible: works with any learning algorithm
- No prior knowledge needed about weak learner
- Provably effective, given weak classifier
- Versatile: can use with data that is textual, numeric, discrete, etc.
- Has been extended to learning problems well beyond binary classification

**Effect of Boosting**

- In the early iterations, boosting primarily reduces bias
- In later iterations, boosting primarily reduces variance (apparently)

**Boosting Conclusions**

- Boosting is a practical tool for classification and other learning problems
  - Grounded in rich theory
  - Performs well experimentally
  - Often (not always!) resistant to overfitting
  - Many applications and extensions
- Many ways to think about why boosting works
  - None is entirely satisfactory
  - Considerable room for further theoretical and experimental work

**Random Forests**

A variant of BAGGING

**Algorithm**

Repeat k times
1. Draw with replacement \(N\) examples, put in train set
2. Build d-tree, but in each recursive call
   A. Choose (w/o replacement) \(i\) features
   B. Choose best of these \(i\) as the root of this (sub)tree
3. Do NOT prune

**More on Random Forests**

- Increasing \(i\)
  - Increases correlation among individual trees (BAD)
  - Also increases accuracy of individual trees (GOOD)
- Can use tuning set to choose good setting for \(i\)
- Overall, random forests
  - Are very fast (e.g., 50K examples, 10 features, 10 trees/min on 1 GHz CPU in 2004)
  - Deal with large \(f\) of features
  - Reduce overfitting substantially
  - Work very well in practice
Stacking

- Apply multiple base learners (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation:
  Meta-L. inputs = Predictions on left-out examples

Error-Correcting Output Coding

- **Motivation:** Applying binary classifiers to multiclass problems
- **Train:** Repeat $L$ times:
  - Form a binary problem by randomly assigning classes to “superclasses” 0 and 1
  - E.g.: A, B, D → 0; C, E → 1
  - Apply binary learner to binary problem
- **Test:**
  - Apply each classifier to test example, forming vector of predictions $\mathbf{P}$
  - Predict class whose vector is closest to $\mathbf{P}$ (Hamming)

Model Ensembles: Summary

- Learn several models and combine them
- **Bagging:** Random resamples
- **Boosting:** Weighted resamples
- **ECOC:** Recode outputs
- **Stacking:** Multiple learners