Machine Learning

- We've studied several supervised learning algorithms
  - Decision Trees
  - K-Nearest Neighbors
  - Naive Bayes
- There are many more
- But we also talked about the problem of "overfitting"

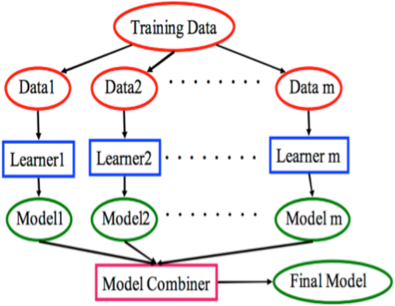
Overfitting

- We might be tempted to improve our classifiers performance by adding more training data, or adding additional features.
- This can lead to overfitting, where our classifier learns unintended patterns in the training set that are not predictive of the test set.
- Often there is a threshold where adding data actually reduces performance.
- But maybe we already have a very large data set...

Other problems

- We also talked about noise, for example incorrectly labeled examples
- You may have encountered this in your homework
- Another issue: representational bias, meaning our algorithm can only represent a subset of all possible hypotheses
- There is theory that says we can't fix all this entirely, but one approach that helps is to build an *ensemble* of classifiers

What is an ensemble?



Ensembles

- Instead of one all-powerful classifier, build a collection of simpler ones
- Recall Bayesian optimal classifier & K-NN: taking a “vote” on most likely class
- Ensembles take a vote of multiple classifiers
- Now multiple classifiers have to err to get an example wrong
- Two approaches:
  - Averaging
  - Boosting

An averaging approach: Bagging

- Idea: resample your data with replacement T times to build T classifiers
- Each classifier focuses on a different subset of the data
- To label new data: Take the majority vote of the learned models

Boosting

- Developed from a theoretical perspective: how well can we perform with a *weak learner* that only needs to reach slightly better than 0.5 training accuracy.
- Idea: Each classifier focuses on a subset of H rather than a subset of D as in bagging
- General approach:
  - Examples are given weights
  - Learn T classifiers, iterating on re-weighted examples that focus the system on examples that the last-learned classifier got wrong.
- The resulting ensemble can represent richer hypotheses than the (original) individual classifiers.

Basic Boosting Algorithm

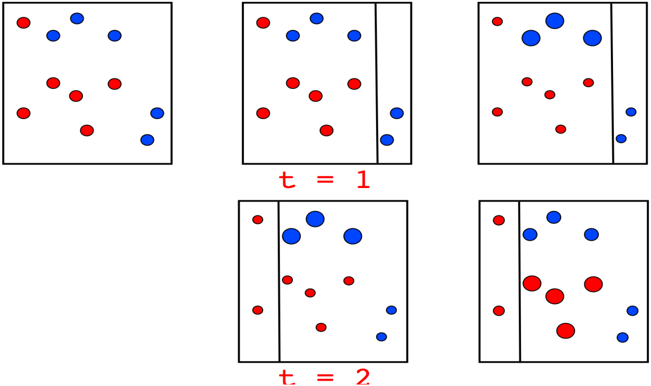
- General loop:
  Set all examples to have equal uniform weights
  Let $H = \{\}$
  For t in range(T) do:
    Learn a hypothesis, h_t from the weighted examples
    Add h_t to H
    Increase the weights of examples h_t classifies incorrectly
- At testing time, each hypothesis gets a weighted vote proportional to its accuracy on its training data.

Learning with weighted Examples

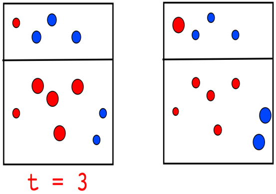
Some classifiers, like Naive Bayes, can handle weighted examples
What about something like decision trees?

- Idea: Replicate examples in the training set proportional to their weights (e.g., 10 replicates of an example with w=0.01 and 100 for one with w=0.1)
- Another way to think about it: select examples from the dataset, using the weights as probability of selection.

Boosting Visualization



Visualization



Let's try some of these ideas

Bagging reminder: resample your data with replacement T times to build T classifiers

To label new data: Take the majority vote of the learned models

Boosting pseudocode

```
Create a set w of N weights, each initially 1/N
For i = 1 to M do:
    build a classifier from D, using weights
    error = sum([w[d] for d in D if d not classified correctly]) / N
    For each correctly classified example:
        w[j] = w[j] * error/(1-error)
    normalize weights so that they add up to 1
    weight of classifier is log((1-error)/error)
```

AdaBoost Advantages

- Fast
- Simple, easy to program
- Can use any off-the-shelf learning algorithm as the basis
- Resistant to overfitting

AdaBoost Disadvantages

- Weak classifiers: too complex leads to overfitting
- Requires a large dataset
- Has been found to be vulnerable to uniform noise

Introduction to the Random Forest algorithm

- Random Forests (RFs) learn an ensemble of decision trees
- Use bagging and random subsets of the attributes & split points
- At each split point, RFs consider only a small fraction of the total number of variables available - reduces the amount of computation. This is referred to as "subset-splitting".
- One of the most accurate off-the-shelf classifiers

Subset splitting

- In Random Forest, only a random subset of the variables, m, is considered at each node.
- A commonly chosen number for m is $\sqrt{m}$
- But in general, $m << d$, where d is the number of attributes

Algorithm

For b = 1 to B:

  Draw a bootstrap sample of size N

  Grow a random forest tree T_b from the bootstrap data:

    select m variables at random

    Pick the best variable/split point

To label: take majority vote for classification

Random Forests

Pros

- Very accurate; more robust to noise than AdaBoost
- Few parameters to tune
- Much more robust to changes in the data - often very little preprocessing of the data needs to be performed, as the data does not need to be normalised and the approach is resilient to outliers
- Also suitable when there are very many input variables and not so many observations
- Helps avoid overfitting

Random Forests

Cons

- Slower to train than many other methods
- Unintuitive output
- Can still overfit particularly noisy data

Summary

- Ensemble techniques very popular and accurate
- Bagging: each classifier focuses on different parts of the sample space
- Boosting: each classifier focuses on different parts of the hypothesis space
- Random Forests: combines the two ideas
- Still not a panacea