COMP90051 Statistical Machine Learning

Semester 2, 2015

Lecturer: Ben Rubinstein

THE UNIVERSITY OF
MELBOURNE

Ensemble Learning

Classifier Combination

To Date ...

- Thus far, we have mostly discussed individual learners and considered each of them in isolation/competition
- We know how to evaluate each classifier's performance (via accuracy, F-measure, etc.) which allows us to choose the best classifier for a dataset *overall*
- Would we find that errors made by the "best" classifier on a dataset are on a proper subset of the instances a worse classifier misclassified? Almost certainly NO!
- *Overall*-worse classifiers, might still be superior on *some* instances

Classifier Combination

- **Classifier combination** (aka. ensemble learning) constructs a set of **base classifiers** from a given set of training data and aggregates the outputs into a single **meta-classifier**
- Motivation 1: the sum of lots of weak classifiers can be at least as good as one strong classifier
- Motivation 2: the sum of a selection of strong classifiers is (usually) at least as good as the best of the base classifiers

Prediction with Combined Classifiers

- The simplest means of classification over multiple base classifiers is simple **voting**:
  - ⋆ for a nominal class set, run multiple base classifiers over the test data and select the class predicted by the most base classifiers (cf. k-NN)
  - ⋆ for a continuous class set, average over the numeric predictions of our base classifiers

Why does Combination Work?

- Intuition: Suppose we trained 25 binary base classifiers, each with high error rate $\epsilon = 0.35$. Assuming the base classifiers are independent and we perform classifier combination by voting, the error rate of the combined classifier is

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

- More generally: Many of today's ensemble learners come with theoretical results, such as reduced variance.

Approaches to Learner Combination

- **Instance manipulation:** generate multiple training datasets through sampling, and train a base classifier over each
- **Feature manipulation:** generate multiple training datasets through different feature subsets, and train a base classifier over each
- **Class label manipulation:** generate multiple training datasets by manipulating the class labels in a reversible manner
- **Algorithm manipulation:** semi-randomly "tweak" internal parameters within a given algorithm to generate multiple base classifiers over a given dataset

Bagging (bootstrap aggregating; Breiman'94)

- Intuition: the more data, the better the performance
  - ⋆ Lower variance (e.g. combination by averaging) $\Rightarrow$ lower MSE
  - ⋆ So how can we get more data out of a fixed training dataset?
- Method: construct "novel" datasets via sampling w replacement
  - ⋆ Generate k datasets, each size N sampled from training data w replacement
  - ⋆ Build base classifier on each constructed dataset; combine predictions via voting

Bagging: Sampling Example

- Original training dataset:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- Bootstrap samples:

  $\{7, 2, 6, 7, 5, 4, 8, 8, 1\ 0\}$ — out-of-sample 3, 9
  $\{1, 3, 8, 0, 3, 5, 8, 0, 1, 9\}$ — out-of-sample 2, 4, 6, 7
  $\{2, 9, 4, 2, 7, 9, 3, 0, 1, 0\}$ — out-of-sample 3, 5, 6, 8

Important Example: Random Forests

- Just bagged trees!
- Algorithm (params: no. trees T, no. features $F \leq d$)
  1. Initialise forest as empty
  2. For $t = 1 \ldots T$
     i. Create new bootstrap sample of training data
     ii. Select random subset of F of the d features
     iii. Train decision tree on bootstrap sample using the F features
     iv. Add tree to forest
  3. Prediction on test instances: majority class of trees' predictions
- Works well in many practical settings

Putting Out-of-Sample Data to Use

- On average only 63.2% of the data will be included per training dataset
- Can use this for error estimate of ensemble
  - ⋆ Evaluate each base classifier on corresponding out-of-sample 36.8% data
  - ⋆ Average these accuracies
- Estimate is unbiased!

Bagging: Reflections

- Simple method based on sampling and voting
- Possibility to parallelise computation of individual base classifiers
- Highly effective over noisy datasets
- Performance is generally significantly better than the base classifiers but never substantially worse
- Improves *unstable* classifiers by reducing variance (no proof), not bias

Boosting

- Intuition: focus attention of base classifiers on examples "hard to classify"

- Method: iteratively change the **distribution** on examples to reflect performance of the classifier on the previous iteration
  - ⋆ start with each training instance having a $\frac{1}{N}$ probability of being included in the sample
  - ⋆ over T iterations, train a classifier and update the weight of each instance according to classifier's ability to classify it
  - ⋆ combine the base classifiers via weighted voting

Boosting: Sampling Example

- Original training dataset:

0	1	2	3	4	5	6	7	8	9

- Boosting samples:

Iteration 1:

7	2	6	7	5	4	8	8	1	0

Iteration 2:

1	3	8	4	3	5	4	0	1	4

Iteration 3:

4	9	4	2	4	4	3	0	1	4

⋮

AdaBoost (Freund & Schapire '96)

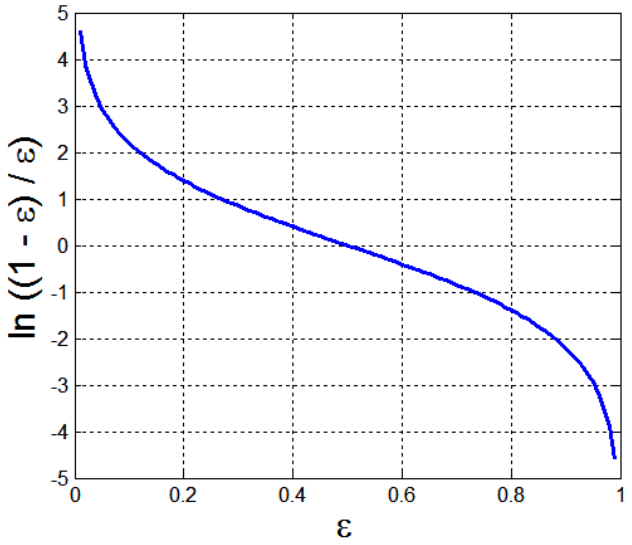
- Initialise example distribution $D_1(i) = 1/n$
- For $t = 1 \ldots T$:
  1. Train base classifier on sample w replacement from D_t
  2. Set confidence $\alpha_t = \frac{1}{2} \log_e \frac{1-\epsilon_t}{\epsilon_t}$ for classifier's error rate ϵ_t
  3. Update example distribution $D_{t+1}(i)$ to be normalised of:

$$D_t(i) \times \begin{cases} \exp(-\alpha_t) \ , & \text{if classifier correct on example } i \\ \exp(\alpha_t) \ , & \text{otherwise} \end{cases}$$

- Classify as majority vote weighted by confidences $\arg\max_y \sum_{t=1}^{T} \alpha_t \delta(C_t(x) = y)$

AdaBoost (cont.)

confidence weights:



- Technicality: Reinitialise example distribution whenever $\epsilon_t > 0.5$
- Base classifiers: often decision stumps or trees, anything "weak"

Boosting: Reflections

- Mathematically complicated but computationally cheap method based on iterative sampling and weighted voting
- More computationally expensive than bagging
- The method has guaranteed performance in the form of error bounds over the training data
- In practical applications, boosting can overfit

Bagging vs. Boosting

Bagging	Boosting
Parallel sampling	Iterative sampling
Simple voting	Weighted voting
Single classification algorithm	Single classification algorithm
Minimise variance	Target "hard" instances
Not prone to overfitting	Prone to overfitting

Stacking

- Basic intuition: "smooth" errors over a range of algorithms with different biases
- Method 1: simple voting

  presupposes the classifiers have equal performance
- Method 2: train a classifier over the outputs of the base classifiers (**meta-classification**)
  - ⋆ Train base- and meta-classifiers using cross-validation
  - ⋆ Simple meta-classifier: linear

Stacking: Reflections

- Mathematically simple but computationally expensive method
- Able to combine heterogeneous classifiers with varying performance
- With care, stacking results in as good or better results than the best of the base classifiers
- Widely seen in applied research; less interest within theoretical circles (esp. statistical learning)

Summary

- What is classifier combination?
- What is bagging and what is the basic thinking behind it?
- What is boosting and what is the basic thinking behind it?
- What is stacking and what is the basic thinking behind it?
- How do bagging and boosting compare?