Lecture 12. Ensemble methods.

COMP90051 Statistical Machine Learning

Semester 2, 2018
Lecturer: Ben Rubinstein

THE UNIVERSITY OF MELBOURNE

This lecture

- Ensemble methods: Hedging your bets!
  - Bagging and random forests
  - Boosting
  - Stacking

art: OpenClipartVectors at pixabay.com (CC0)

Why "one true" model?

- Thus far, we have discussed individual models and considered each of them in isolation/competition
- We know how to evaluate each model's performance (via accuracy, F-measure, etc.) which allows us to choose the best model for a dataset *overall*
- But overall doesn't imply per-example performance
  * Overall best model: likely makes errors on some instances!
  * Overall-worst model: could be superior on some instances!
- Ensembles let us use multiple models together!

Panel of experts

- Consider a panel of 3 experts that make a classification decision independently. Each expert makes a mistake with probability 0.3. The consensus decision is by majority vote.

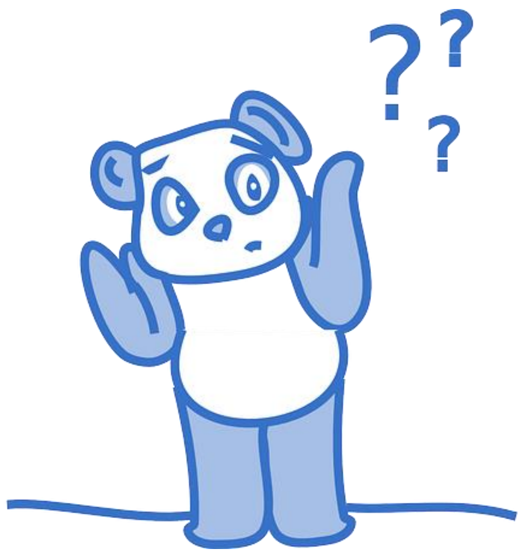
What is the probability of a mistake in the consensus decision?

$0.189 = 3 \times 0.3 \times 0.3 \times 0.7$

$0.790 = 0.7 + 0.3 \times 0.3$

$0.216 = 0.3^3 + 3 \times 0.63$

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**



art: OpenClipartVectors at pixabay.com (CC0)

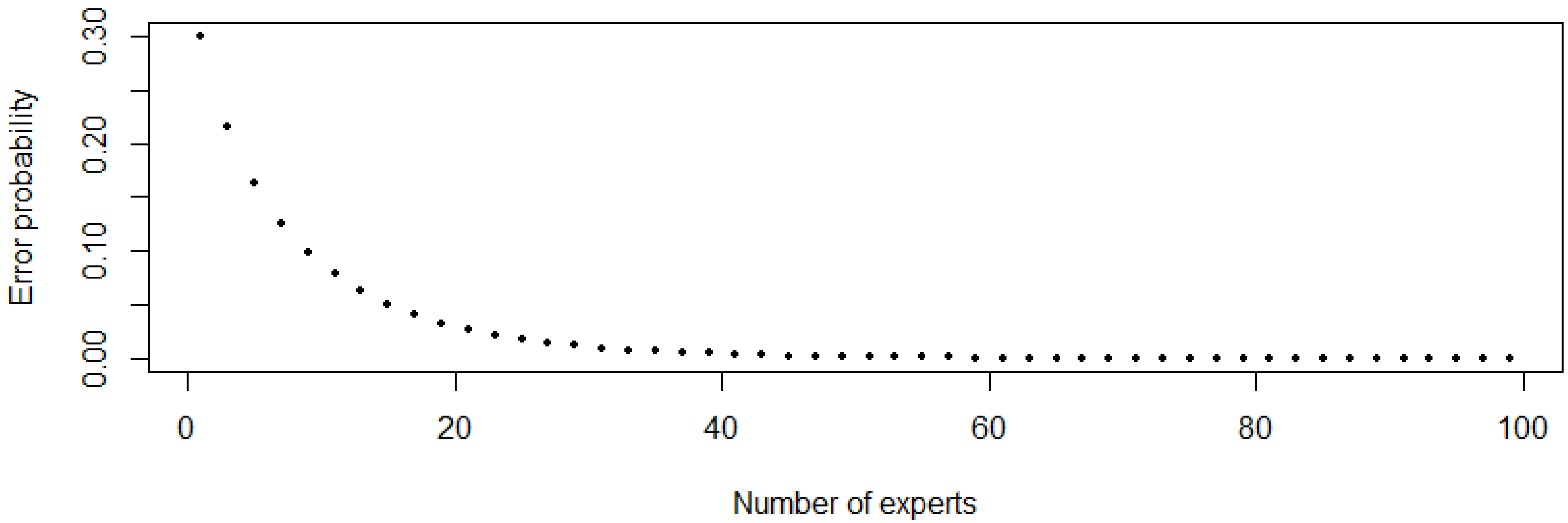
Panel of experts (cont.)

- Setup: Independent mistakes, each probability *p*
- Distribution of #mistakes of *n* experts is Binom(n,p)

$$\Pr(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

- Majority vote errs when *at least n/2 experts err*

$$\Pr(panel\ error) = \sum_{k=n/2}^{n} \binom{n}{k} p^k (1-p)^{n-k}$$



Combining models

- Model combination (aka. ensemble learning) constructs a set of base models (aka base learners) from given training set and aggregates the outputs into a single meta-model (ensemble)
  - Classification via (weighted) majority vote
  - Regression via (weighed) averaging
  - More generally: *meta-model = f(base models)*

How to generate multiple learners from a single training dataset?

- Recall bias-variance trade-off:

$$\mathbb{E}\left[l\left(\mathcal{Y}, \hat{f}(\boldsymbol{x}_0)\right)\right] = \left(\mathbb{E}[\mathcal{Y}] - \mathbb{E}[\hat{f}]\right)^2 + Var[\hat{f}] + Var[\mathcal{Y}]$$

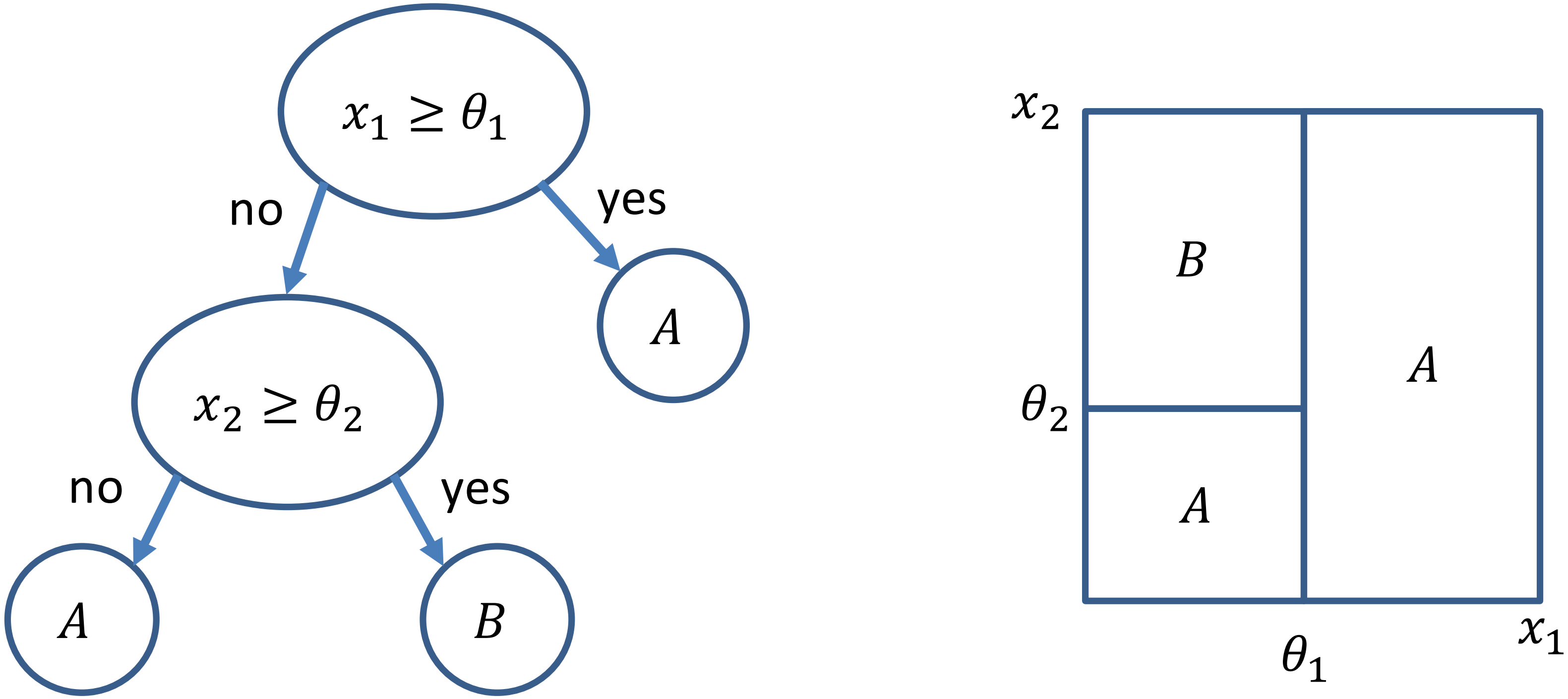
Test error = $(\text{bias})^2$ + variance + irreducible error

- Averaging k *independent* and identically distributed predictions reduces variance: $Var[\hat{f}_{avg}] = \frac{1}{k} Var[\hat{f}]$

Bagging (bootstrap aggregating; *Breiman'94*)

- Method: construct "near-independent" datasets via sampling with replacement
  - Generate k datasets, each size n sampled from n training data with replacement – bootstrap samples
  - Build base classifier on each constructed dataset
  - Aggregate predictions via voting/averaging
- 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 $3, 5, 6, 8$

Refresher on decision trees



- Training criterion: Purity of each final partition
- Optimisation: Heuristic greedy iterative approach
- Model complexity is defined by the depth of the tree
- Deep trees: Very fine tuned to specific data → high variance, low bias
- Shallow trees: Crude approximation → low variance, high bias

Bagging example: Random forest

- Just bagged trees!
- Algorithm (parameters: #trees k, #features $l \leq m$)
  1. Initialise forest as empty
  2. For $c = 1 \dots k$
     a) Create new bootstrap sample of training data
     b) Select random subset of l of the m features
     c) Train decision tree on bootstrap sample using the l features
     d) Add tree to forest
  3. Making predictions via majority vote or averaging
- Works *extremely* well in many practical settings

Putting out-of-sample data to use

- At each round, a particular training example has a probability of $\left(1 - \frac{1}{n}\right)$ of not being selected
  - Thus probability of being left out is $\left(1 - \frac{1}{n}\right)^n$
  - For large n, this probability approaches $e^{-1} \approx 0.368$
  - On average only 63.2% of data included per bootstrap sample
- Can use this for *independent* error estimate of ensemble
  - Safe like cross-validation, but on overlapping sub-samples!
  - Evaluate each base classifier on its out-of-sample 36.8%
  - Average these evaluations → Evaluation of ensemble!

Bagging: Reflections

- Simple method based on sampling and voting
- Possibility to parallelise computation of individual base classifiers
- Highly effective over noisy datasets
- Performance is often significantly better than (simple) base classifiers, never substantially worse
- Improves *unstable* classifiers by reducing variance

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 $1/n$ probability of being included in the sample
  - Over k 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

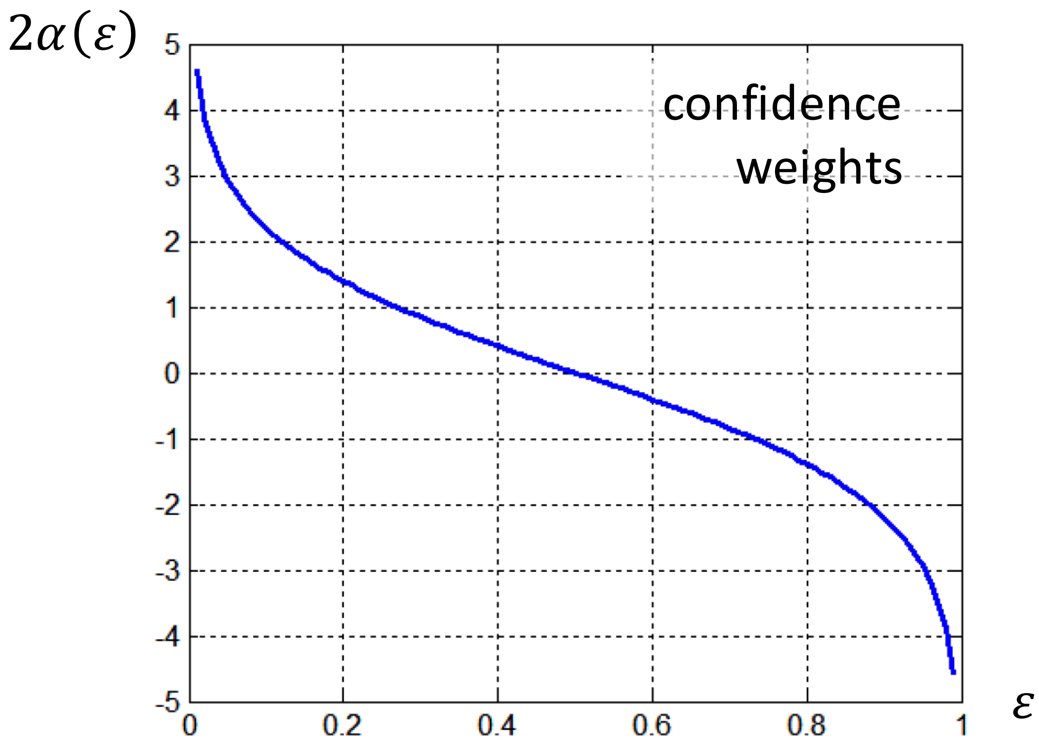
Boosting: Sampling example

- Original training dataset:
  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Boosting samples:
  Iteration 1: $\{7, \mathbf{2}, 6, 7, 5, 4, 8, 8, 1\ 0\}$
  Suppose that example 2 was misclassified
  Iteration 2: $\{1, 3, 8, \mathbf{2}, 3, 5, \mathbf{2}, 0, 1, 9\}$
  Suppose that example 2 was misclassified still
  Iteration 3: $\{\mathbf{2}, 9, \mathbf{2}, \mathbf{2}, 7, 9, 3, \mathbf{2}, 1, 0\}$

Boosting Example: AdaBoost

1. Initialise example distribution $P_1(i) = 1/n, i = 1, \dots, n$
2. For $c = 1 \dots k$
   a) Train base classifier A_c on sample with replacement from P_c
   b) Set confidence $\alpha_c = \frac{1}{2} \ln \frac{1-\varepsilon_c}{\varepsilon_c}$ for classifier's error rate ε_c
   c) Update example distribution to be normalised of:
   $$P_{c+1}(i) \propto P_c(i) \times \begin{cases} \exp(-\alpha_c), & if\ A_c(i) = y_i \\ \exp(\alpha_c), & otherwise \end{cases}$$
3. Classify as majority vote weighted by confidences
   $\arg\max_y \sum_{c=1}^{k} \alpha_t \delta(A_c(\boldsymbol{x}) = y)$

AdaBoost (cont.)



- Technicality: Reinitialise example distribution whenever $\varepsilon_t > 0.5$
- Base learners: often decision stumps or trees, anything "weak"
  * A *decision stump* is a decision tree with one splitting node

Boosting: Reflections

- 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
Minimise variance	Target “hard” instances
Simple voting	Weighted voting
Classification or regression	Classification or regression
Not prone to overfitting	Prone to overfitting (unless base learners are simple)

Stacking

- Intuition: "smooth" errors over a range of algorithms with different biases
- Method: train a meta-model over the outputs of the base learners
  - Train base- and meta-learners using cross-validation
  - Simple meta-classifier: logistic regression
- Generalisation of bagging and boosting

Stacking: Reflections

- Compare this to ANNs and basis expansion
- 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

This lecture

- Ensemble methods: Hedging your bets!
  - Bagging and random forests
  - Boosting
  - Stacking

art: OpenClipartVectors at pixabay.com (CC0)