Lecture 12. Ensemble methods. Interim Revision

COMP90051 Statistical Machine Learning

Semester 2, 2017
Lecturer: Andrey Kan

This lecture

- Ensemble methods
  - Bagging and random forest
  - Boosting and stacking
- Frequentist supervised learning
  - Interim summary
- Discussion

art: OpenClipartVectors at pixabay.com (CC0)

Ensemble Methods

Overview of model combination approaches

Choosing a model

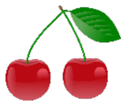
- Thus far, we have mostly discussed individual models and considered each of them in isolation/competition
- We know how to evaluate each model's performance (via accuracy, F-measure, etc.) which allows us to choose the best model for a dataset *overall*
- This "best" model is still likely to make errors on some instances.
- Overall-worse models, might still be superior on some instances!

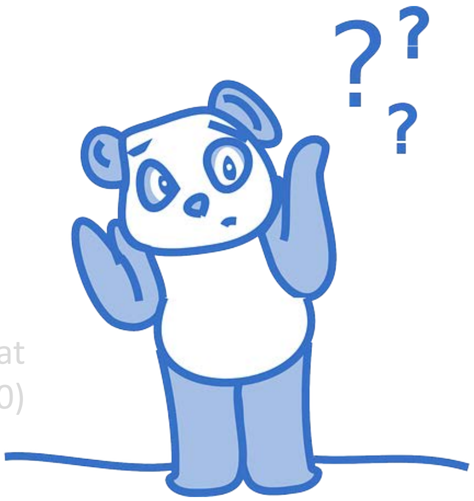
Panel of experts

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

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

$0.7 + 0.3 \times 0.3 = 0.79$


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



Combining models

- Model combination (aka. ***ensemble learning***) constructs a set of ***base models*** (aka ***learners***) from a given set of training data and aggregates the outputs into a single ***meta-model***
  - 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:

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

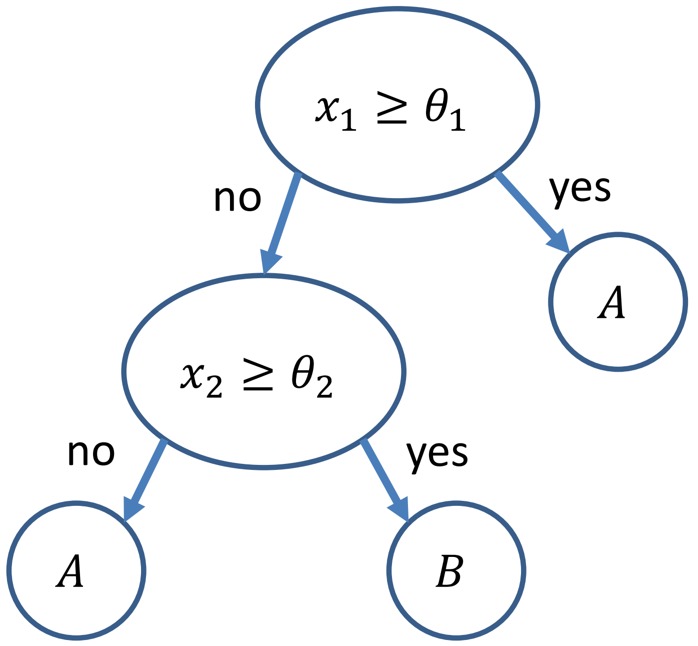
Test error = $(\text{bias})^2$ + variance + irreducible error

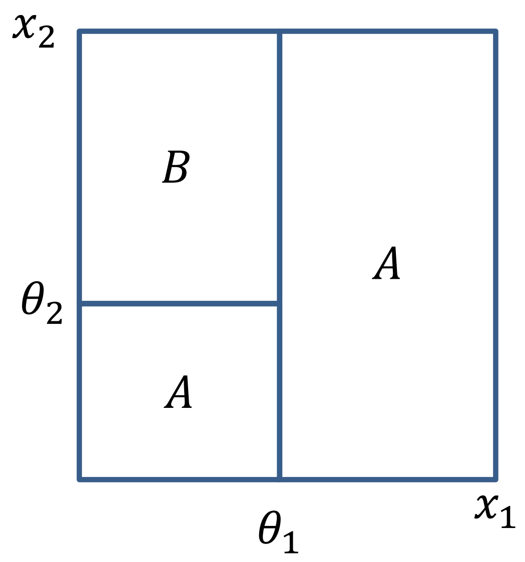
- Averaging k <u>*independent*</u> and identically distributed predictions reduces variance: $Var\left[\hat{f}_{avg}\right] = \frac{1}{k} Var\left[\hat{f}\right]$

Bagging (bootstrap aggregating; *Breiman'94*)

- Method: construct "novel" datasets via sampling with replacement
  - Generate k datasets, each size n sampled from training data with replacement
  - Build base classifier on each constructed dataset
  - Combine 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 a 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 well in many practical settings

Putting out-of-sample data to use

- At each round, a particular training example has a probability of $(1 - \frac{1}{n})$ of not being selected
  - Thus probability of being left out is $\left(1 - \frac{1}{n}\right)^n$
  - For large n, this probability approaches $e^{-1} = 0.368$
  - On average only 63.2% of the data will be included per training dataset
- Can use this for error estimate of ensemble
  - Essentially cross-validation
  - Evaluate each base classifier on corresponding out-of-sample 36.8% data
  - Average these accuracies

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

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: $\{1, 3, 8, \mathbf{2}, 3, 5, \mathbf{2}, 0, 1, 9\}$

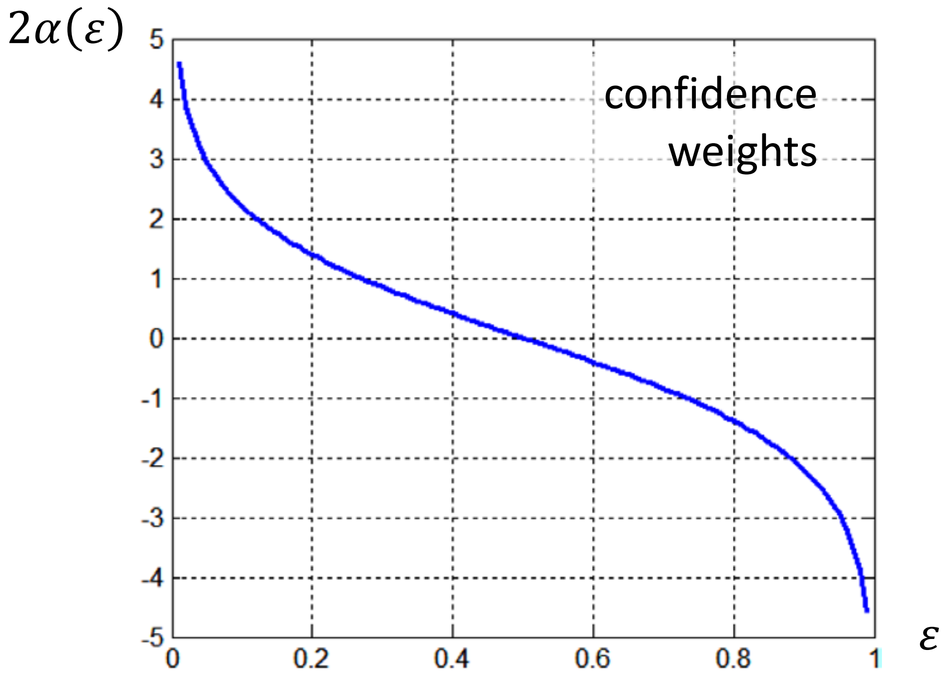
  Suppose that example 2 was misclassified still

  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

Supervised Learning

Interim summary of frequentist supervised learning methods covered so far

Supervised learning*

1. Assume a model (e.g., linear model)
   * Denote parameters of the model as $\boldsymbol{\theta}$
   * Model predictions are $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$
2. Choose a way to measure discrepancy between predictions and training data
   * E.g., sum of squared residuals $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$
3. Training $=$ parameter estimation $=$ optimisation

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} L(data, \boldsymbol{\theta})$$

*This is the setup of what's called *frequentist supervised learning*. A different view on parameter estimation/training will be presented later in the subject.

Supervised learning methods (1/3)

- Linear Regression (Galton, Pearson)
  - Model: $\mathcal{Y} = \boldsymbol{x}'\boldsymbol{w} + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
  - Loss function: Squared loss
  - Optimisation: Analytic solution (the normal equations)
  - Notes: Can also be optimised iteratively
- Logistic Regression (Cox)
  - Model: $p(y|\boldsymbol{x}) = Bernoulli\left(y|\theta(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{x}'\boldsymbol{w})}\right)$
  - Loss function: Cross-entropy (*aka* log loss)
  - Optimisation: Iterative, 2nd order method
- Perceptron (Rosenblatt)
  - Model: Label is based on sign of $w_0 + \boldsymbol{w}'\boldsymbol{x}$
  - Loss function: Perceptron loss
  - Optimisation: Stochastic gradient descent
  - Notes: Provable convergence for linearly separable data

Supervised learning methods (2/3)

- Artificial Neural Networks (Hinton, LeCun)
  - Model: Defined by network topology
  - Loss function: Varies
  - Optimisation: Variations of gradient descent
  - Notes: Backpropagation used to compute partial derivatives
- Support Vector Machines (Vapnik)
  - Model: Label is based on sign of $b + \boldsymbol{w}'\boldsymbol{x}$
  - Loss function: Hard margin SVM loss; hinge loss
  - Optimisation: Quadratic Programming
  - Notes: Specialised optimisation algorithms (e.g., SMO, chunking)
- Random Forest (Breiman)
  - Model: Average of decision trees (combination of piece-wise constant models)
  - Loss function: Cross-entropy (*aka* log loss); squared loss
  - Optimisation: Greedy growth of each tree
  - Notes: This is an example of model averaging

Supervised learning methods (3/3)

- The Next Super-Method (You)
  - *(that is, if you really need a new one)*
  - What are the aims of the method? What is the scope of the method? Intended use? Assumptions?
  - Model: Analytically or algorithmically defined?
  - Loss function: What is the relevant goodness criterion?
  - Optimisation: Is there an efficient method for training?

Basis expansion

- All Methods
  - Manually craft a feature space transformation (e.g., polynomial basis, RBF basis), before using the method
- Artificial Neural Networks
  - Earlier layers can be viewed as transformation
  - Topology needs to be pre-defined, but weights are learned from data
- Linear Regression, Logistic Regression, Perceptron, Support vector machines
  - *Name a common aspect of these methods*
  - Kernelise and use implicit transformation by choosing a kernel
- Ensemble Methods, including Random Forest
  - Base models as feature space transformation (learned)

Regularisation

- Can be used for various purposes
  - Add resilience to (nearly) collinear features
  - Introduce prior knowledge into the process of learning
  - **Control model complexity**
- Ability to generalise reflected in test error
  - Simple models: underfit, high bias, low variance
  - Complex models: overfit, low bias, high variance
- Method 1: Analytically, by adding a data-independent term to the objective function, e.g.:
  - Ridge regression
  - Lasso
- Method 2: Algorithmically, by not allowing the model to “fine-tune”, e.g.:
  - Early sopping in ANN
  - Weights sharing in CNN
  - Restricting tree depth in Random Forests

What is Machine Learning?

This lecture

- Ensemble methods
  - Bagging and random forest
  - Boosting and stacking
- Frequentist supervised learning
  - Interim summary
- Discussion

art: OpenClipartVectors at pixabay.com (CC0)