COMP30027 Machine Learning
Classifier Combination

Semester 1, 2019

Jeremy Nicholson & Tim Baldwin & Karin Verspoor

Lecture Outline

To Date ... I

- Thus far, we have discussed individual classification algorithms and considered each of them in isolation/competition
- We have discussed ways of comparing the performance of individual classifiers over a given dataset/task, which allows us to choose the "dataset optimal" classifier
- If we were to carry out error analysis of multiple classifiers over a given dataset, would we find that the errors made by better-performing classifiers were over a proper subset of instances that worse-performing classifiers similarly misclassified? Almost certainly NO!

To Date ... II

- When evaluating, we only get one shot at classifying a given test instance and are stuck with the bias inherent in a given algorithm

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 combination of lots of weak classifiers can be at least as good as one strong classifier
- Motivation 2: the combination of a selection of strong classifiers is (usually) at least as good as the best of the base classifiers

Source(s): Tan et al. [2006, p277–80]

Why does Combination Work? I

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

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

- ...And if the classifiers aren't independent?

Classification 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

Approaches to Classifier 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

Source(s): Tan et al. [2006, p277–80]

Lecture Outline

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 using nested cross validation to reduce bias

Source(s): Witten and Frank [2005, p332–4]

Stacking example

- Given training dataset (X, y):
  - Train SVM
  - Train Naive Bayes
  - Train Decision Tree
- Discard (or keep) X, add new attributes for each instance:
  - predictions (labels) of the classifiers above
  - other data as available (NB scores, SVM $wx + b$, etc.)
- Train meta-classifier (usually Logistic Regression)

Stacking: Reflections

- Mathematically simple but computationally expensive method
- Able to combine heterogeneous classifiers with varying performance
- Generally, 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)

Lecture Outline

Bagging I

- Basic intuition: the more data, the better the performance (lower the variance), so how can we get ever more data out of a fixed training dataset?
- Method: construct "novel" datasets through a combination of random sampling and replacement
  - Randomly sample the original dataset N times, with replacement
  - Thus, we get a new dataset of the same size, where any individual instance is *absent* with probability $(1 - \frac{1}{N})^N$
  - construct k random datasets for k base classifiers, and arrive at prediction via voting

Bagging: Sampling Example

- Original training dataset:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

- Bootstrap samples:

7	*2*	*6*	*7*	*5*	*4*	*8*	*8*	*1*	*10*

1	*3*	*8*	*10*	*3*	*5*	*8*	*10*	*1*	*9*

2	*9*	*4*	*2*	*7*	*9*	*3*	*10*	*1*	*10*

$\vdots$

Bagging: Classification Algorithm

- The same (weak) classification algorithm is used throughout
- As bagging is aimed towards minimising variance through sampling, the algorithm should be unstable (= high-variance) ... e.g.?

Bagging: Reflections

- Simple method based on sampling and voting
- Possibility to parallelise computation of individual base classifiers
- Highly effective over noisy datasets (outliers may vanish)
- Performance is generally significantly better than the base classifiers and only occasionally substantially worse

Lecture Outline

Random Tree

A "Random Tree" is a Decision Tree where:

- At each node, only *some* of the possible attributes are considered
- For example, a fixed proportion τ of all of the attributes, except the ones used earlier in the tree
- Attempts to control for unhelpful attributes in the feature set
- Much faster to build than a "deterministic" Decision Tree, but increases model variance

Random Forests

A "Random Forest" is:

- An ensemble of Random Trees (many trees = forest)
- Each tree is built using a different Bagged training dataset
- As with Bagging the combined classification is via voting
- The idea behind them is to minimise overall model variance, without introducing (combined) model bias

Source(s): Breiman [2001], Tan et al. [2006, p290–293]

Random Forests (cont.)

- Hyperparameters:
  - number of trees B (can be tuned, e.g. based on "out-of-bag" error rate)
  - feature sub-sample size (e.g. $\lfloor \log_2 |F| + 1 \rfloor$)
- Interpretation:
  - logic behind predictions on individual instances can be tediously followed through the various trees
  - logic behind overall model: ???

Practical Properties of Random Forests

- Generally a very strong performer
- Embarrassingly parallelisable
- Surprisingly efficient
- Robust to overfitting
- Interpretability sacrificed

Lecture Outline

Boosting

- Basic intuition: tune base classifiers to focus on the "hard to classify" instances
- Method: iteratively change the distribution and **weights** of training instances to reflect the 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 whether it is correctly classified
  - combine the base classifiers via weighted voting

Source(s): Tan et al. [2006, p285–90]

Boosting: Sampling Example

- Original training dataset:

1	2	3	4	5	6	7	8	9	10

- Boosting samples:

Iteration 1:	7	2	6	7	5	4	8	8	1	10
Iteration 2:	1	3	8	4	3	5	4	10	1	4
Iteration 3:	4	9	4	2	4	4	3	10	1	4

⋮

Boosting Example: AdaBoost I

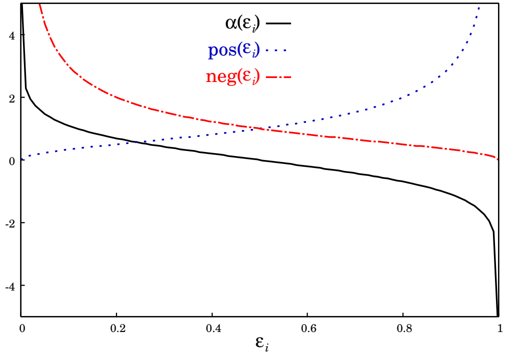
- Base classifiers: $C_1, C_2, ..., C_T$
- Training instances $\{(x_j, y_j) | j = 1, 2, ..., N\}$
- Initial instance weights $\{w_j^{(0)} = \frac{1}{N} | j = 1, 2, ..., N\}$
- Error rate for C_i:

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j^{(i)} \, \delta(C_i(x_j) \neq y_j)$$

Boosting Example: AdaBoost II

- "Importance" of C_i (i.e. the weight associated with the classifiers' votes):

$$\alpha_i = \frac{1}{2}\log_e \frac{1-\epsilon_i}{\epsilon_i}$$



Boosting Example: AdaBoost III

- Instance weights for $i > 0$:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

Boosting Example: AdaBoost IV

- Continue iterating for $i = 1, 2, ..., T$, but reinitialise the instance weights whenever $\epsilon_i > 0.5$
- Classification:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \, \delta(C_j(x) = y)$$

- Base classification algorithm: decision stumps (OneR) or decision trees

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
- Interesting effect with convergence of the error rate over the training vs. test data
- In practical applications, boosting has the tendency to overfit

Bagging/RF vs. Boosting

Bagging/RF	Boosting
Parallel sampling	Iterative sampling
Simple voting	Weighted voting
Single classification algorithm	Single classification algorithm
Minimise variance	Minimise (instance) bias
Not prone to overfitting	Prone to overfitting

Lecture Outline

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?

References I

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, USA, second edition, 2005.