

# Lecture 20: Ensemble Learning

---

**COMP90049**

Semester 1, 2024

Ting Dang, CIS

Copyright @ University of Melbourne 2024. All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the author.

Acknowledgement: Lea Frermann, Jeremy Nicholson, Tim Baldwin & Karin Verspoor



## So far...

- individual classification algorithms in isolation
- choose the "optimal" classifier by comparing the performance of individual classifiers over a given dataset/task
- When evaluating, we only get one shot at classifying a given test instance and are stuck with the bias inherent in a given algorithm

## Today

- Ensembles: combining multiple (weak/unstable) models into one strong model!



## **Aside: (Non)-linear and (non)-parametric classification**

---

### Linear classifiers

- Naive Bayes
- Logistic Regression
- Perceptron

... because their **decision boundary** is a linear function of the input  $x$ .  
(There's still a non-linear activation function, so  $y$  is not a linear function of  $x$ ).

### Non-linear classifiers

- Multi-layer perceptron (with non-linear activations)
- K-Nearest Neighbors
- Decision trees

... because their **decision boundary** is *\*not\** a linear function of the input  $x$ .  
They can learn more complex decision boundaries.



## Aside II: parametric vs. non-parametric models

**Warning:** these terms are ambiguous and several definitions exist. We'll adopt the following.

### Parametric Models

- Naive Bayes, Logistic Regression, Multi-layer perceptron, ...

... because they have a **constant number of parameters**, irrespective of the **amount of training data**. We can write down the model  $y = f(x; \theta)$  which holds true no matter what  $x$ . We fit parameters to a **given model**.

### Non-parametric models

- K-Nearest Neighbors, Decision trees, ...

... because the parameters grow with the training data and are **possibly infinite**. We **learn our model directly from the data**.

- Discuss: what's 'non-parametric' about KNN?
- Discuss: what's 'non-parametric' about Decision Trees?



**Now, on to ensembles**

---

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

# Ensembles: Toy example

- When does ensemble learning work?
  - the base classifiers should not make the same mistakes
  - the base classifiers are reasonably accurate

	$t_1$	$t_2$	$t_3$
$C_1$	✓	✓	x
$C_2$	x	✓	✓
$C_3$	✓	x	✓
$C^*$	✓	✓	✓

	$t_1$	$t_2$	$t_3$
$C_1$	✓	✓	x
$C_2$	✓	✓	x
$C_3$	✓	✓	x
$C^*$	✓	✓	x

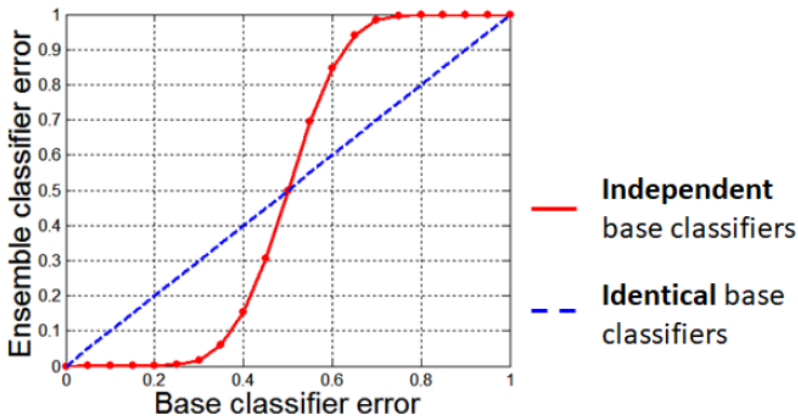
	$t_1$	$t_2$	$t_3$
$C_1$	✓	x	x
$C_2$	x	✓	x
$C_3$	x	x	✓
$C^*$	x	x	x





# Ensembles: Error rate of base classifiers

- When does ensemble learning work?



**Let's assume that:**

- We have a set of 25 binary base classifiers
- Each has an error rate of  $\epsilon = 0.35$
- The base classifiers are independent (that's usually false)
- We perform classifier combination by voting

**The error rate of the combined classifier is:**

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



- The simplest means of classification over multiple base classifiers is simple **voting**:
  - **Classification**: run multiple base classifiers over the test data and select the class predicted by the most base classifiers (e.g. k-NN)
  - **Regression**: average over the numeric predictions of our base classifiers

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

## Stacking

---

- **Intuition:** “smooth” errors over a range of algorithms with different biases
- **Simple Voting:** generate multiple training datasets through different feature subsets, and train a base classifier over each dataset
  - presupposes the classifiers have equal performance
- **Meta Classification:** train a classifier over the outputs of the base classifiers
  - train using nested cross validation to reduce bias



- **Level 0:** Given training dataset  $(X, y)$ :
  - Train Neural Network
  - Train Naive Bayes
  - Train Decision Tree
  - ...
- Discard (or keep)  $X$ , **add new attributes** for each instance:
  - predictions of the classifiers above
  - other data as available (NB scores etc.)
- **Level 1:** Train meta-classifier. Usually Logistic Regression or Neural Network

## Nested Cross-validation



- 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)

## Bagging

---

- **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** (bootstrap)
  - 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: Example

- Original 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
---	---	---	---	---	---	---	----	---	----

⋮

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

- 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

## **Bagging - Random Forest**

---

A **Random Tree** is a Decision Tree where:

- At each node, only some of the possible attributes are considered
- For example, a fixed proportion  $\tau$  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

This is an instance of **Feature Manipulation**.







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

This is an instance of **Instance Manipulation**.

## Hyperparameters:

- number of trees  $B$  (can be tuned, e.g. based on “out-of-bag” error rate)
- feature sub-sample size (e.g.  $(\log |F| + 1)$ )

## 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

## Boosting

---

- **Intuition:** tune base classifiers to focus on the “hard to classify” instances
- **Approach:** 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 probability of  $\frac{1}{N}$  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**

# Boosting: Example

- Original dataset:

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

- Boosting samples:

<i>Iteration 1:</i>	7	2	6	7	5	4	8	8	1	10
---------------------	---	---	---	---	---	---	---	---	---	----

<i>Iteration 2:</i>	1	3	8	4	3	5	4	10	1	4
---------------------	---	---	---	---	---	---	---	----	---	---

<i>Iteration 3:</i>	4	9	4	2	4	4	3	10	1	4
---------------------	---	---	---	---	---	---	---	----	---	---

⋮

## **Adaptive Boosting: AdaBoost**

---

**AdaBoost** (Adaptive Boosting) is a *sequential* ensembling algorithm.

## Basic idea:

- Base classifier:  $C_0$
- Training instances  $(x_j, y_j) | j = 1, 2, \dots, N$
- Initial instance weights  $w_j^{(0)} = \frac{1}{N} | j = 1, 2, \dots, N$
- In iteration  $i$ :
  - Construct classifier  $C_i$  and compute **error rate**  $\epsilon_i$

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

- Use  $\epsilon_i$  to compute the **classifier weight**  $\alpha_i$  (importance of  $C_i$ )
- Use  $\alpha_i$  to update **instance weights**
- Add weighted  $C_i$  to ensemble

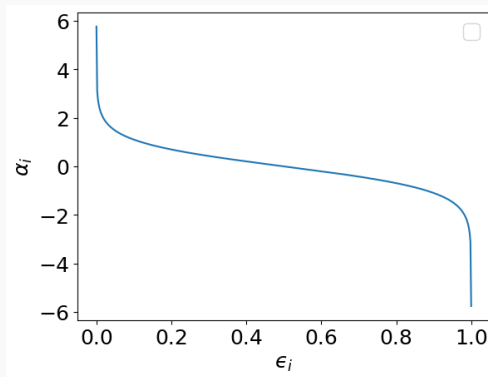
**Output:** Weighted set of base classifiers:  $\{(\alpha_1, C_1), (\alpha_2, C_2), \dots, (\alpha_T, C_T)\}$





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

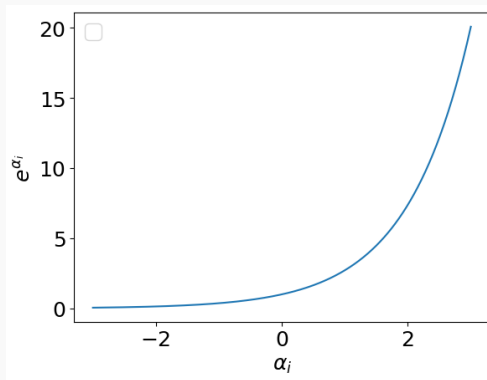
$$\alpha_i = \frac{1}{2} \log_e \frac{1 - \epsilon_i}{\epsilon_i}$$



## AdaBoost III: Updating $w$

**Weights** for instance  $j$  ( $i > 0$ ):

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



(recall that the  $\alpha_i$ s are always positive)

- Continue iterating for  $i = 1, 2, \dots, T$ , but re-initialise the instance weights whenever  $\epsilon_i > 0.5$
- As long as each base classifier is better than random, convergence to a stronger model is **guaranteed**
- **Classification:** weighted voting

$$C^*(x) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$$

- **Typical base classifiers:** decision stumps (OneR) or decision trees. (Possibly the world's favorite classifier!)



## Boosting: Summary

- 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

- Parallel sampling
- Simple voting
- Single classification algorithm
- Minimise variance
- Not prone to overfitting

## Boosting

- Iterative sampling
- Weighted voting
- Single classification algorithm
- Minimise (instance) bias
- Prone to overfitting

## 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?

- 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.