Lecture 1: Introduction and Decision Trees

COSC470

Brendan McCane

Department of Computer Science, University of Otago

Teaching Team

- Brendan McCane (L1-3)
- Michael Albert (L4, L8, L12)
- Lech Szymanski (L5-7)
- Steven Mills (L9-11)

Learning Outcomes

By the end of this paper you should have:

1. an understanding of, an ability to use, and an ability to implement classical machine learning techniques
2. an understanding of how AI agents play games and the most recent techniques for game playing
3. an understanding of the basics of machine learning theory and how it applies to the recent success of deep learning
4. an ability to implement a deep learning model for a given task
5. an understanding of the fundamentals of image recognition
6. an understanding of deep convolutional neural networks and an ability to use and implement such models
7. an ability to conduct AI experiments and successfully communicate the results
8. an ability to find, read and understand recent research in the area
9. an appreciation for the ethical problems associated with AI

Lecture Outline

1. Intro to classification and ML; decision trees (BM)
2. Ensemble Methods: AdaBoost and Random Forests (BM)
3. Support Vector Machines (BM)
4. History of game playing, basic ideas, themes. (MA)
5. Basics of Learning Theory (and a bit of deep vs. shallow) (LS)
6. Deep Learning: classification/convnets/regularisation techniques etc. (LS)
7. Deep reinforcement learning: Atari games/LSTMs/DQNs (LS)
8. The alphago breakthrough (and alphazero) (MA)
9. Bag of words and 'traditional' recognition (SM)
10. Faces (Viola Jones detection and Eigenface recognition) (SM)
11. Convnets and image-based object recognition + (SM)
12. What next? [Are two player perfect information games just a solved case now? What are the more serious application cases?] (MA)
13. Wrap-up, discussion of AI ethics. (Guest lecturer: Ali Knott)

Assessment

This paper is 100% internally assessed. Each assignment is worth 25% of the paper and there is one assignment per lecturer/lecture block.

- A1: end of week 5, Aug 10 (AdaBoost)
- A2: end of week 8, Sep 7 (CNN)
- A3: end of week 11, Sep 28 (Report on AI techniques for games)
- A4: end of week 13, Oct 12 (Face recognition)

Academic Integrity and Academic Misconduct

Academic integrity means being honest in your studying and assessments. It is the basis for ethical decision-making and behaviour in an academic context. Academic integrity is informed by the values of honesty, trust, responsibility, fairness, respect and courage. Students are expected to be aware of, and act in accordance with, the University's Academic Integrity Policy.

Academic Misconduct, such as plagiarism or cheating, is a breach of Academic Integrity and is taken very seriously by the University. Types of misconduct include plagiarism, copying, unauthorised collaboration, taking unauthorised material into a test or exam, impersonation, and assisting someone else's misconduct. A more extensive list of the types of academic misconduct and associated processes and penalties is available in the University's Student Academic Misconduct Procedures.

Academic Integrity Continued

It is your responsibility to be aware of and use acceptable academic practices when completing your assessments. To access the information in the Academic Integrity Policy and learn more, please visit the Universitys Academic Integrity website at `www.otago.ac.nz/study/academicintegrity` or ask at the Student Learning Centre or Library. If you have any questions, ask Brendan, Michael, Lech or Steven.

- Academic Integrity Policy (`www.otago.ac.nz/administration/policies/otago116838.html`)
- Student Academic Misconduct Procedures (`http://www.otago.ac.nz/administration/policies/otago116850.html`)

An example ML task

Class	Att 1	Att 2	Att 3	Att 4	Att 5	Att 6
2	*	*	*	*	*	2
1	2	*	*	*	*	1
1	1	2	*	*	*	1
1	1	1	*	*	*	1
1	1	3	2	2	*	1
1	*	*	*	*	4	1
2	1	4	*	*	1	1
2	1	4	*	*	2	1
2	1	4	*	*	3	1
2	1	3	1	1	1	1
2	1	3	1	1	2	1
2	1	3	1	2	1	1
2	1	3	1	2	2	1
1	1	3	1	1	3	1
2	1	3	1	2	3	1

Land the space shuttle

Auto-land?	Stability	Error	Sign	Wind	Magnitude	Visibility
Auto	*	*	*	*	*	2
Manual	2	*	*	*	*	1
Manual	1	2	*	*	*	1
Manual	1	1	*	*	*	1
Manual	1	3	2	2	*	1
Manual	*	*	*	*	4	1
Auto	1	4	*	*	1	1
Auto	1	4	*	*	2	1
Auto	1	4	*	*	3	1
Auto	1	3	1	1	1	1
Auto	1	3	1	1	2	1
Auto	1	3	1	2	1	1
Auto	1	3	1	2	2	1
Manual	1	3	1	1	3	1
Auto	1	3	1	2	3	1

What is Machine Learning?

Machine learning is the process of learning a function, structure or process from input training data, such that the function, structure or process produces desirable output given data *it hasn't seen before*.

What is Machine Learning?

There are several dimensions on which to categorise machine learning:

- classification versus regression problems
- supervised versus unsupervised problems
- active versus passive learners
- categorical or numerical attributes
- symbolic versus sub-symbolic learners
- immediate or delayed rewards

In this paper we will focus mostly on supervised classification problems, mostly with numerical attributes, and mostly sub-symbolic learners. But we will also look at active learners with delayed rewards.

Some Notation

Lower case bold indicates a vector: $\mathbf{x} \in \mathbb{R}^d$.

Vectors are used to represent an input sample (usually $\mathbf{x}$) with d dimensions or attributes or features.

Upper case bold indicates a matrix: $\mathbf{X} \in \mathbb{R}^{n \times d}$.

Matrices are used to represent n input samples (usually $\mathbf{X}$) with n rows and d columns.

A target or label is represented by the vector $\mathbf{y} \in \mathbb{R}^l$, where l is the dimensionality of the target vector. For two-class problems, $l = 1$.

Targets are represented by the matrix $\mathbf{Y} \in \mathbb{R}^{n \times l}$.

Upper case, but non-bold characters (X, Y) are used to indicate underlying populations from which we get the samples: $\mathbf{X}, \mathbf{Y} \subseteq X, Y$.

The Supervised Classification Problem

Given a population X, Y, use the data $\mathbf{X}, \mathbf{Y}$, to build a function, $f : \mathbb{R}^d \to \mathbb{R}^l$, so that, as much as possible, for $\mathbf{x}, \mathbf{y} \in X, Y$:

$$f(\mathbf{x}) = \mathbf{y}$$

In other words, we want to build some function that can tell us what the class or label is for every possible sample in the population of interest.

The difficulty is that we usually do not have all possible samples, and indeed, the number of possible samples might be infinite or at least extremely large. So we have to work with the data we do have: $\mathbf{X}, \mathbf{Y}$.

The simplest classifier - nearest neighbours

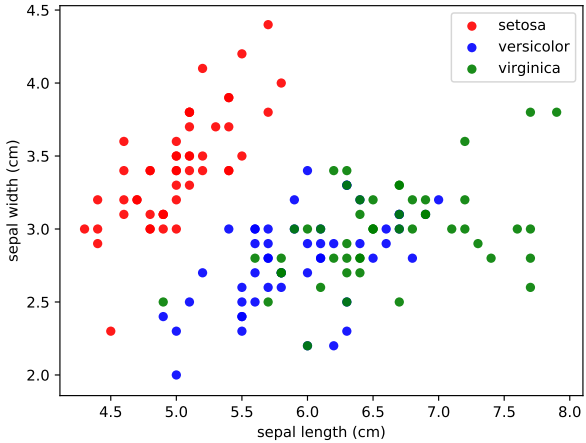
For low-dimensional problems, nearest neighbours (k-NN) is very accurate, but not very efficient.

The basic idea is simple. Just remember all the training data, and use the label of the nearest point (for 1-NN, or nearest k points for k-NN) in feature space as the label for the new point. That is, f is defined as:

$$f(\mathbf{x}) = \mathbf{Y}_i \text{ where } i = \underset{0 \leq j < n}{\arg\min} ||\mathbf{x} - \mathbf{X}_j||$$

where $\mathbf{X}_i$, $\mathbf{Y}_i$ indicate the i^{th} row of $\mathbf{X}, \mathbf{Y}$, and $||\mathbf{x}||$ is some vector norm of $\mathbf{x}$.

An Example - Iris recognition



Nearest Neighbours - in Python

It can be hard at first to get used to mathematical notation, so here is function f implemented in python:

```
def f_nn(x, X, Y):
    n,d = X.shape
    closest_dist = np.inf
    closest_label = -1
    for i in xrange(n):
        diff = X[i,:]-x
        dist_squared = np.dot(diff, diff)
        if dist_squared<closest_dist:
            closest_dist = dist_squared
            closest_label = Y[i,:]

    return np.sqrt(closest_dist), closest_label
```


Problems with Nearest Neighbours?

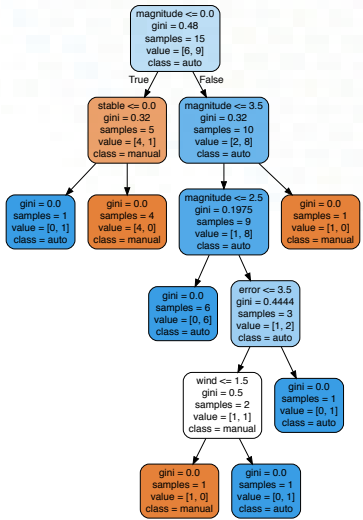
Questions to ask when evaluating classification methods:

- How long does it take to train?
- How long does it take to classify?
- How much memory does it use?
- How accurate is it (are there any accuracy guarantees)?
- How does it scale with dimension?

Decision Trees

Decision trees are very simple:

- A decision tree is (usually) a binary tree
- Each node splits the data into two halves (we'll discuss how in the following)
- The tree is grown in a greedy fashion as building an optimal tree is NP-complete
- Keep splitting nodes until some criterion is met. E.g. until a node has only one class in it
- Prune the tree using some cost criterion



How to choose split points?

The most common types of splits are axis-aligned (on one attribute). This results in rectangular regions of feature space.

It is usually feasible to test every possible split point on every attribute.

Let $Q(\mathbf{X}, \mathbf{Y})$ be the impurity measure for node data (lower is better).

Splitting Algorithm

Function `GrowTree`$(\mathbf{X}, \mathbf{Y})$
- m = new Tree$(\mathbf{X}, \mathbf{Y})$
- **if** *Stop*$(\mathbf{X}, \mathbf{Y})$ **then**
  - m.class = most common element of $\mathbf{Y}$
  - **return** *m*
- **end**
- $n, d = \mathbf{X}.shape$
- $Q^* = \infty$
- **for** $i = 0$ **to** d **do**
  - sort $\mathbf{X}, \mathbf{Y}$ along dimension i
  - **for** $j = 0$ **to** $n - 1$ **do**
    - split $\mathbf{X}, \mathbf{Y}$ between $\mathbf{X}_{j,i}$ and $\mathbf{X}_{j,i+1}$
    - call results $\mathbf{X}_{\text{left}}, \mathbf{X}_{\text{right}}$
    - $q = Q(\mathbf{X}_{\text{left}}, \mathbf{Y}_{\text{left}}) + Q(\mathbf{X}_{\text{right}}, \mathbf{Y}_{\text{right}})$
    - **if** $q < Q^*$ **then**
      - bestSplit $= \mathbf{X}_{\text{left}}, \mathbf{X}_{\text{right}}, \mathbf{Y}_{\text{left}}, \mathbf{Y}_{\text{right}}$
    - **end**
  - **end**
- **end**
- m.left = GrowTree$(\mathbf{X}_{\text{left}}, \mathbf{Y}_{\text{left}})$
- m.right = GrowTree$(\mathbf{X}_{\text{right}}, \mathbf{Y}_{\text{right}})$
- **return** *m*

end

Efficiency

- How long does it take to train?
- How long does it take to classify?
- How much memory does it use?
- How accurate is it (are there any accuracy guarantees)?
- How does it scale with dimension?

What about Q?

There are a few common options. First, some more notation.

Let the number of classes be K, and let $0 \leq k < K$ represent a particular class.

Let $\hat{p}_k(\mathbf{X}, \mathbf{Y})$ be the proportion of class k in data $\mathbf{X}, \mathbf{Y}$. In mathematical notation we would write:

$$\hat{p}_k(\mathbf{X}, \mathbf{Y}) = \frac{1}{n} \sum_{i=0}^{n} I(\mathbf{Y}_i = k)$$

where I is called the indicator function which is 1 if the condition is true, and 0 otherwise.

What about Q?

Misclassification error:

$$Q_{\text{Mis}} = 1 - \hat{p}_k(\mathbf{X}, \mathbf{Y})$$

Gini index:

$$Q_G = \sum_{k=0}^{K-1} \hat{p}_k(\mathbf{X}, \mathbf{Y})(1 - \hat{p}_k(\mathbf{X}, \mathbf{Y}))$$

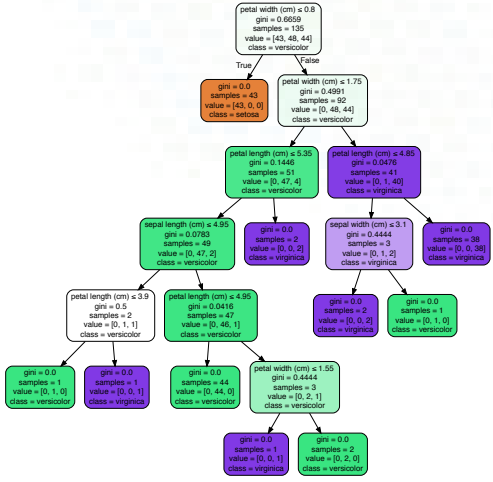
Cross-entropy:

$$Q_{CE} = -\sum_{k=0}^{K-1} \hat{p}_k(\mathbf{X}, \mathbf{Y}) \log \hat{p}_k(\mathbf{X}, \mathbf{Y})$$

Some examples ...

On the iris data

- The performance of a classifier on the training data gives almost no information about how good the classifier is.
- One strategy is to use cross-validation: Train on a random subset of the data, then test on a different subset.
- For the iris data with 20 rounds, and leaving out 15 samples per round for testing, the average error (with std.dev) is $\approx 0.05 \pm 0.05$.



Overfitting

If we build a "pure" tree, then there is a very good chance that we have overfit the data.
Their are two ways of avoiding overfitting with decision trees:

1. stop building the tree early (early stopping)
2. build a pure tree, then prune.

Early stopping is easy to implement but requires knowledge of the data to work well. Pruning tends to just work.

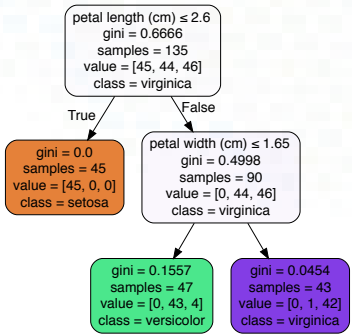


Figure: Early stopping with minimum samples per leaf = 40. Error $\approx 0.05 \pm 0.06$

Cost-complexity Pruning

Let $\mathbf{X}_m, \mathbf{Y}_m$ be the data associated with node m of a tree T, n_m be the number of training samples in node m, and $|T|$ be the number of nodes in tree T.

The idea is to generate a sequence of progressively smaller trees until we are left with just the root.

Cost-complexity Pruning

At each stage choose an internal node and make it a leaf (collapsing all its children). Add the resulting tree, T' to the list.

The node to choose at each stage is the one that minimizes:

$$\frac{1}{|T'|} \sum_{m \in T'} n_m Q(\mathbf{X}_m, \mathbf{Y}_m)$$

Choose the tree from this sequence that minimises the following cost:

$$C_\alpha(T) = \sum_{m=1}^{|T|} n_m Q(\mathbf{X}_m, \mathbf{Y}_m) + \alpha |T|$$

α is called a *regularisation* parameter and is used to trade-off size of tree with accuracy on training data.

Homework

1. Install anaconda on your machine (this is a python distribution)
2. Install sklearn via anaconda (conda install sklearn)
3. Run through the quick tutorial on `http://scikit-learn.org/stable/modules/tree.html#tree` (make sure it works on your installation).
4. Implement your own decision tree (from algorithm on page 20) and compare with the sklearn version.
5. Implement early stopping and cost-complexity pruning and compare the two on several datasets.