CO395 - Introduction to Machine Learning (70050)

Week 2 (Introduction to ML)

Week 3 (Instance-based Learning + Decision Trees)

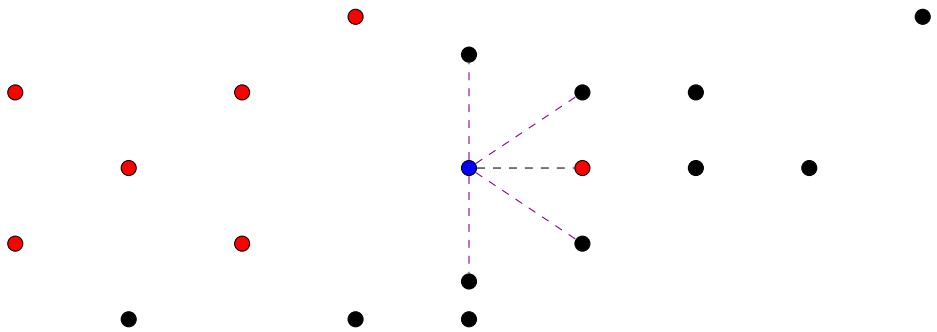
The **k Nearest Neighbours (k-NN)** classifier is classified as a **lazy learner**. A lazy learner stores all the training examples in the data set, and postpone any processing until a request is made (such as a prediction). On the other hand, **decision trees** are classified as a **eager learner**. An eager learner will attempt to construct a general target decision function, which is prepared prior to a query being made.

Classification with Instance-based Learning

The concept behind instance-based learning is that we will use samples in a training data set in order to make inference on a query.

The **Nearest Neighbour** classifier is a specific example, where it classifies a test instance to the label of the nearest training instance, where nearest is subject to some distance metric. This is a **non-parametric model**, which means it naturally emerges from the training set. Note in the example below, an issue with this is that it can be sensitive to noise, as it would classify the blue point to be red, as it is the closest instance in the training set, even though it's more likely to be black - it is very sensitive to noise, and can **overfit** to the training data.

On the other hand, if we consider the **k Nearest Neighbours**, highlighted by the lines in violet, we get the class to be black, as we have 4 against 1. Usually, we need k to be odd, to ensure a winner for the decision task.



Increasing k will give the classifier have a smoother decision boundary (higher bias), and less sensitive to training data (lower variance). Choosing k is dependant on the dataset, normally with a validation dataset.

The distance metric can be defined in many different ways, including the ℓ_1, ℓ_2 and ℓ_∞-norms as seen in **CO233**. Other metrics exist such as the **Mahalanobis distance** for non-isotropic spaces, typically used for Gaussian distributions, or the **Hamming distance** for binary strings.

Another variation is the **Distance Weighted k-NN**. For example, we may not want to trust neighbours which are further away, such as in the example below.

The idea is that we add weights to each neighbour (depending on distance), typically a higher weight for closer neighbours. We then assign the class based on which class has the largest sum. This metric, $w^{(i)}$, is any measure favouring the votes of nearby neighbours, such as;

- inverse of distance

$$w^{(i)} = \frac{1}{d(x^{(i)}, x^{(q)})}$$

- Gaussian distribution

$$w^{(i)} = \frac{1}{\sqrt{2\pi}} e^{-\frac{d\left(x^{(i)}, x^{(q)}\right)^2}{2}}$$

The value of k is less important in the weighted case, as distant examples won't greatly affect classification. If $k = N$, where N is the size of the training set, it is a global method, otherwise it is a local method (only considering the samples close by). This method is also more robust to noisy training data, however it can be slow for large datasets.

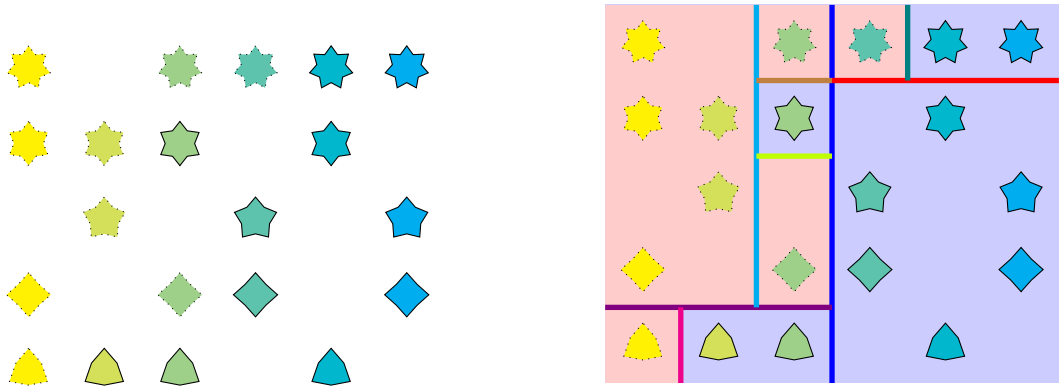
As this method relies on distance metrics, it may not work well if using all features in high dimensional spaces. If these features are irrelevant, instances in the same class may be far from each other. One solution to this is to weight features differently.

k-NN can also be used for regression, either by computing the mean value across k nearest neighbours (which leads to a very rough curve), or by using locally weighted regression, which computes the weighted mean value across k nearest neighbours, leading to a smoother curve.

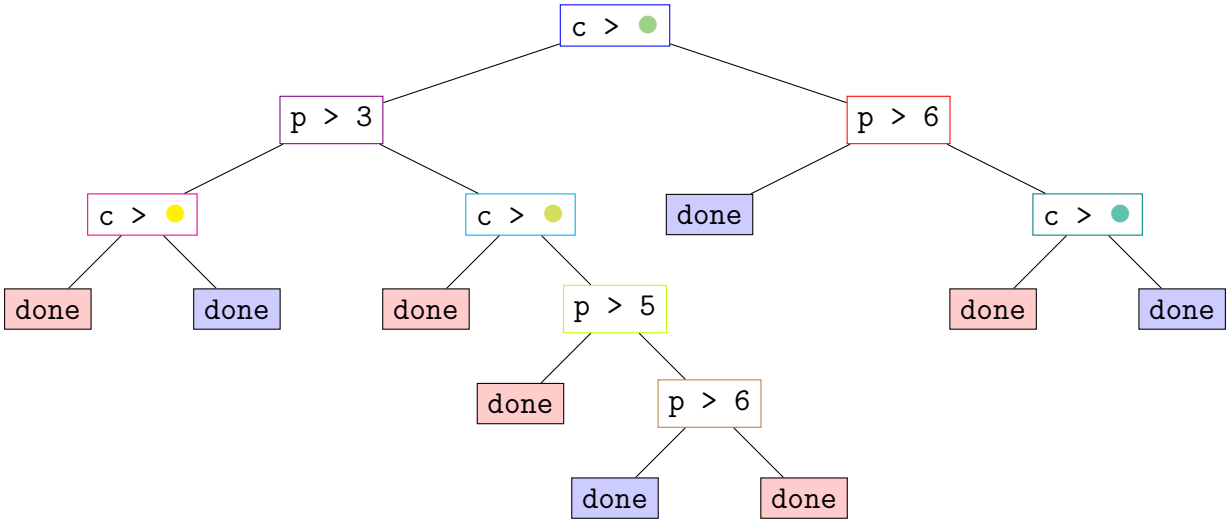
Classification with Decision Trees

Decisions trees are the principal of focusing on a subset or single feature of each sample and then make a decision whether it's true or false (for each feature), and repeat this process to finer decisions until we manage to classify the sample that we want to check.

In decision trees, we learn a succession of linear decision boundaries that we can use to eventually correctly classify samples.



In the example above, we repeatedly choose divisions that result in the fewest number of errors, until we are able to classify everything. This results in the following decision tree, when we are using the attributes of colour and number of points. For brevity, the left branch is the `false` branch, `p` means points, and `c` means colour.



Decision trees are a method of approximating discrete classification functions, by representing them as a tree (a set of if-then rules). The general algorithm (ID3) for constructing a decision tree is as follows;

1. search for the optimal splitting rule on training data
2. split data according to rule
3. repeat 1 and 2 on each subset until each subset is pure (only containing a single class)

How to select the 'optimal' split rule

Intuitively, we want to partition the datasets such that they are more pure than the original set. To do this, we have several metrics;

- **Information gain** ID3, C4.5

  quantifies the reduction of **entropy**

- **Gini impurity** CART

  if we randomly select a point in the feature space and randomly classify it according to the class label distribution, what is our probability of getting it incorrect?

- **Variance reduction** CART

  mostly used for regression trees, with a continuous target variable

To do this, we need to understand information entropy. Entropy is a measure of uncertainty of a random variable. It can also be seen as the average amount of information needed to define a random state / variable. If something has low entropy, it's predictable, and vice versa for high entropy.

Imagine we have two boxes, with something stored in one of the two, with an equal probability in each. To be fully certain, we need a single bit of information, if it's in the left box, the bit is `0`, otherwise (if it's in the right box), it's `1`. Similarly, if we have four boxes, with a uniform distribution, we would need 4 bits to encode the 4 states. In general;

$$
\begin{aligned}
2^B &= K \text{ states} \\
B &= \log_2(K) \\
I(x) &= \log_2(K) && \text{amount of information to determine the state of a random variable} \\
P(x) &= \frac{1}{K} && \Rightarrow \\
K &= \frac{1}{K} && \Rightarrow \\
I(x) &= -\log_2(P(x))
\end{aligned}
$$

As such, we can say;

$$I(x = \text{box}_1) = I(x = \text{box}_2) = I(x = \text{box}_3) = I(x = \text{box}_4) = -\log_2(P(x)) = 2 \text{ bits}$$

However, assume a non-uniform distribution, with the probabilities being 97%, 1%, 1%, and 1% respectively. If we were told it was in box 1, we do not get a lot of new information (low entropy); however if we were told it was in one of the other three, we high entropy (represents very important information).

$$\begin{aligned} I(x = \text{box}_1) &= -\log_2(0.97) \\ &\approx 0.0439 \text{ bits} \\ I(x = \text{box}_2) &= -\log_2(0.1) \\ &\approx 6.6439 \text{ bits} \end{aligned}$$

Entropy is defined as the average amount of information;

$$H(X) = -\sum_{k}^{K} P(x_k) \log_2(P(x_k))$$

In our example, we therefore have;

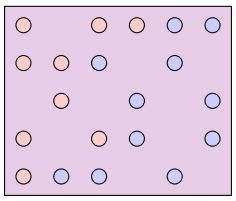
$$H(X) = -(0.97 \cdot \log_2(0.97) + 0.01 \cdot \log_2(0.01) + 0.01 \cdot \log_2(0.01) + 0.01 \cdot \log_2(0.01)) \approx 0.2419 \text{ bits}$$

We therefore need, on average, less information to know where the key is (compared to the uniform distribution).

For continuous entropy, we can use the probability density function $f(x)$ - this is imperfect (it can have negative values), but is still often used in Deep Learning.;

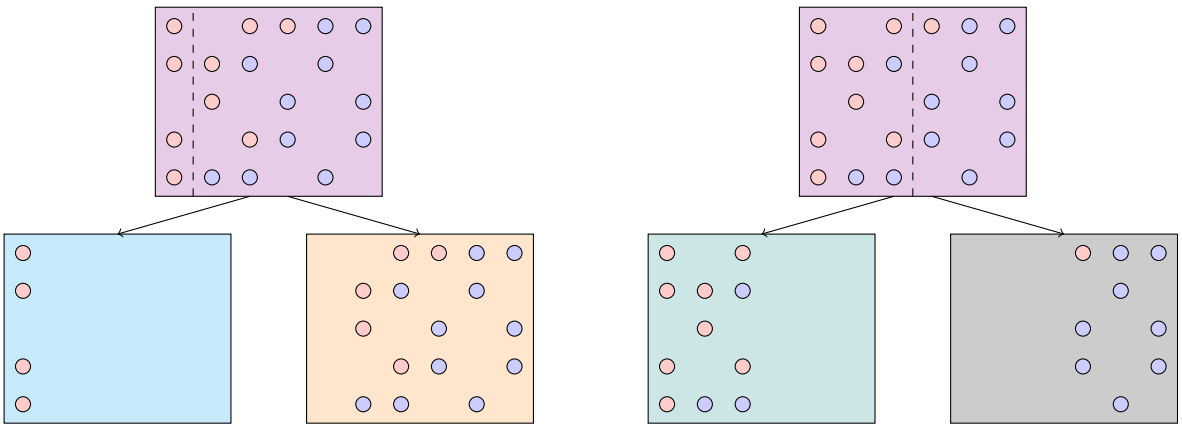
$$H(X) = -\int_{x} f(x) \log_2(f(x)) \, \mathrm{d}x$$

Consider the following example;



$$\begin{aligned} P(\bullet) &= \frac{11}{20} \\ P(\bullet) &= \frac{9}{20} \\ H(\blacksquare) &= -\left(\frac{11}{20} \cdot \log_2\left(\frac{11}{20}\right) + \frac{9}{20} \cdot \log_2\left(\frac{9}{20}\right)\right) \\ &\approx 0.9928 \end{aligned}$$

An entropy value close to 1 would indicate a maximum amount of information needed.



$$
\begin{aligned}
H(\blacksquare) &= 0 \\
H(\blacksquare) &\approx 0.896 \\
H(\{\blacksquare, \blacksquare\}) &\approx \frac{4}{20} \cdot 0 + \frac{16}{20} \cdot 0.896 \\
&\approx 0.7168 \\
H(\blacksquare) - H(\{\blacksquare, \blacksquare\}) &\approx 0.276 \qquad \text{information gain}
\end{aligned}
$$

$$
\begin{aligned}
H(\blacksquare) &\approx 0.8454 \\
H(\blacksquare) &\approx 0.5033 \\
H(\{\blacksquare, \blacksquare\}) &\approx \frac{11}{20} \cdot 0.8454 + \frac{9}{20} \cdot 0.5033 \\
&\approx 0.6915 \\
H(\blacksquare) - H(\{\blacksquare, \blacksquare\}) &\approx 0.3013 \qquad \text{information gain}
\end{aligned}
$$

As the second split has the larger information gain, that is the one we will end up selecting (and generally we want to split to maximise information gain). A formulation of this is as follows;

$$
IG(\text{dataset}, \text{subsets}) = H(\text{dataset}) - \sum_{S \in \text{subsets}} \frac{\mid S \mid}{\mid \text{dataset} \mid} H(S)
$$

$$
\mid \text{dataset} \mid = \sum_{S \in \text{subsets}} \mid S \mid
$$

We can have the following types of input;

- **ordered values**
  - attribute and split point
  - for each attribute, sort the values and consider split points between two examples with different classes
- **categorical / symbolic values**
  - search for the most informative feature and create as many branches as there are values for this feature

Worked example for construction decision tree

Skipped, as this is basically done for the coursework.

Summary and other considerations with decision tree

Note that in general, if we have real-valued attributes, we will end up with a binary tree, with an attribute and threshold at each node. On the other hand, if we have categorical values, we can end up with a **multiway tree**.

Decision trees will **overfit**, like with many machine learning algorithms. This means the algorithm will take into account every sample in the dataset, to the point where it picks up the noise in the dataset. On the other hand, we have an underfitted algorithm, which has low variance and high bias (in contrast).

In decision trees, to deal with overfitting, we can employ the following strategies;

- **early stopping**

  basically stop the algorithm when a condition is met, rather than when the subset is pure (such as maximum depth of tree, or a minimum number of examples in the subset)

- **pruning** will be covered more next week

  1. identify internal nodes connected to only leaf nodes
  2. turn each into a leaf node (with the majority class label)
  3. if the validation accuracy of the pruned tree is greater, we keep it, and then repeat the process until no other pruning can improve the accuracy

To test this, we can reserve part of the dataset for training, and another part for validation. This is called **cross-validation**.

Another approach is to use a random forest. This involves training multiple decision trees, each with a subset of the training dataset, with a random subset of the features, and therefore each focuses on one subset of the features. We then take the majority vote by each of the decision trees as the final outcome.

Decision trees can also be used for regression (**regression trees**). Instead of class labels, each leaf node predicts a real-valued number.

Week 4 (Machine Learning Evaluation)

Week 5 (Artificial Neural Networks I)

Week 6 (Artificial Neural Networks II)

Week 7 (Unsupervised Learning)

Week 8 (Genetic Algorithms)