Gradient Descent

$$\theta \leftarrow \theta - \eta \nabla f(\theta)$$

- η: learning rate/step size
- $\nabla = [\frac{\partial}{\partial \theta_i}]$: gradient; vector of partial derivatives

Distance

Manhattan - L1

Euclidean - L2

Cosine

Jaccard

Hamming

K-Nearest Neighbours

Epsilon Smoothing

Laplace Smoothing

Naive Bayes

Decision Tree

Entropy

- entropy for discrete random variable x without possible outcomes $1, ..., n$:

$$H(x) = -\sum_{i=1}^{n} P(i) \log_2 P(i)$$

Mean Information

- Mean information: m attribute values of x
  - $H(x_i)$: entropy of class distribution for instances at node x_i

- $P(x_i)$: proportion of instances at sub-node x_i

$$\text{mean-info}(x_1, ..., x_m) = \sum_{i=1}^{m} P(x_i)H(x_i)$$

Information gain

$$IG(R_A|R) = H(R) - \text{mean-info}(R_A)$$

- when using IG as split criterion, choose the attribute with the highest IG
- biased towards attributes with many values

Split Information

- Split information: entropy of a given split: evenness of distribution of instances to attribute values

$$SI(R_A|R) = H(R_A) = -\sum_{i=1}^{m} P(x_i) \log_2 P(x_i)$$

Gain Ratio

$$GR(R_A|R) = \frac{IG(R_A|R)}{SI(R_A|R)}$$

- when using GR as split criterion, choose the attribute with the highest GR
- discourages selection of attributes with many uniformly distributed values

ID3

Recursive function:

```
function id3(root):
  if all instances of same class, or other stopping criterion met:
      stop
  else:
      select a new attribute to use to partition node instances (IG or
         GR)
      create a branch for each distinct attribute value
      partition root instances by value
      for each leaf node leaf_i:
          id3(leaf_i)
```

Unsupervised Learning

k-means

```
initialise k seeds as cluster centroids
repeat
  compute distance of all points to cluster centroids
  assign points to the closest cluster
  recompute cluster centroid
until clusters don't change
```

Agglomerative clustering

```
initialise all instances as individual clusters
while (# clusters) > 1
  compute triangular elements of proximity matrix between each cluster
     centroid
  locate the smallest proximity: cluster these instances
  compute updated cluster centroid (typically group average)
```

Evaluation

Cluster - entropy

Cluster - purity

Evaluation

Perceptron

Forward pass

$$\hat{y} = f(\theta^T x)$$

- f may be step, in which case:

$$f(z) = \begin{cases} 1 : z > 0 \\ -1 : otherwise \end{cases}$$

Perceptron update rule

$$\theta \leftarrow \theta + \eta(y^i - \hat{y}^i)x$$

Neural Network update

Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

Backpropagation Algorithm

- design Neural Net
- initialise θ
- one epoch: repeat for each training instance
- forward pass: work from left to right
  - activation for node i of layer l:
    $$a_i^l = \sigma({\theta_i^l}^T x)$$
  - for output node: $\hat{y}$ = activation of last node
- compute error $E = \frac{1}{2}(y - \hat{y})^2$
- backpropagate:
  - compute δ_i^l for node i of layer l:

$$\delta_i^l = \sigma'(z_i^l)(y - \hat{y}) \text{ : final layer}$$

- use $\hat{y} =$ output activation - i.e. look at gradient, and the deviation in activation

$$\delta_i^l = \sum_j \sigma'(z_i^l)\theta_{ij}^{l+1}\delta_j^{l+1} \text{ : hidden layer}$$

- find weighted sum of all δs from the nodes of the next layer, weighted by θ between those nodes

- compute $\Delta\theta$ for all nodes, in any order. For node i in layer l:

$$\Delta\theta_i^l = -\eta\nabla E(\theta) = \eta\delta_i^l a_i^{l-1}$$

- *i.e. multiply learning rate, η, the node's δ, and the node's input a_i^{l-1}*

- update all θs at once. For node i in layer l

$$\theta_i^l \leftarrow \theta_i^l + \Delta\theta_i^l$$

- continue until stop criteria reached