Machine Learning Methods

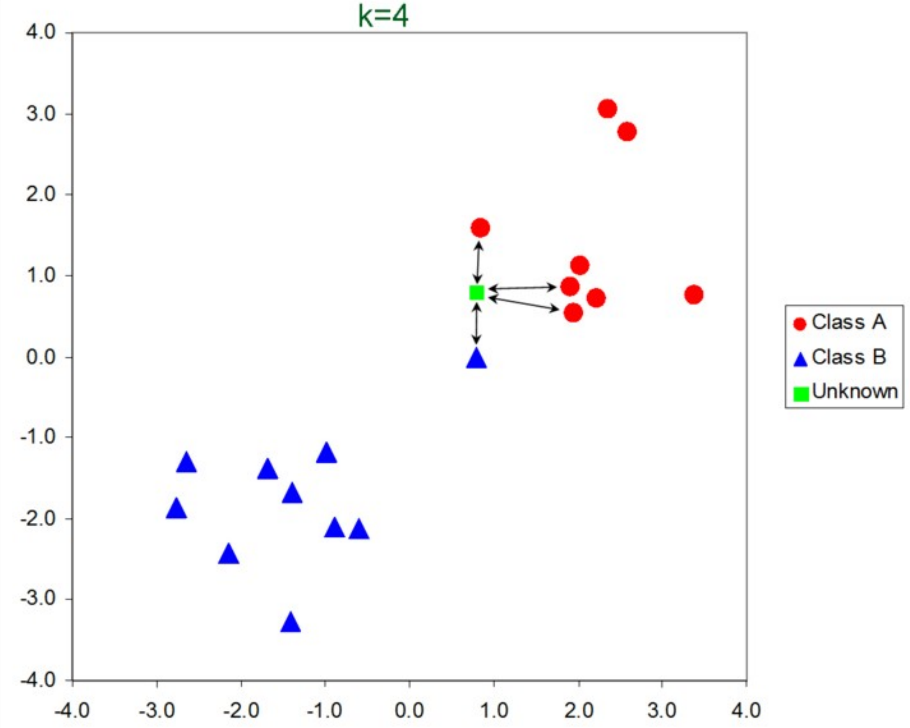
Yeganeh Jalalpour

Brute Force

- Collect training instances in a bag.
- Pick matching training instance from bag, and classify target identically
- Problems:
  - no/multiple matches
  - huge bag
  - slow classification

Minimum Distance Voting (k-Nearest Neighbor)

- Help with some of the problems of brute force
- Pick *k* "closest" instances from bag
  - Metric for boolean features is usually "Hamming Distance"
  - $H(v1, v2) = sum[i](v1[i] \oplus v2[i])$
  - Example:
    - $111 \oplus 101 = 010$ and $H(111,101) = 1$
- Vote the instances



Peterson, L. (2009). K-nearest neighbor. *Scholarpedia Journal.*, *4*(2), 1883.
http://www.scholarpedia.org/article/File:Knn_sample_plot.png

Naïve Bayesian learning

- (We've already looked at this)
- Binary setting: Count the number of occurrences of each feature in positive and negative setting
- Compare underestimates of probabilities using products
- Take logs to turn products into sums
- Use m-separation to get accurate products

Decision Trees and ID3

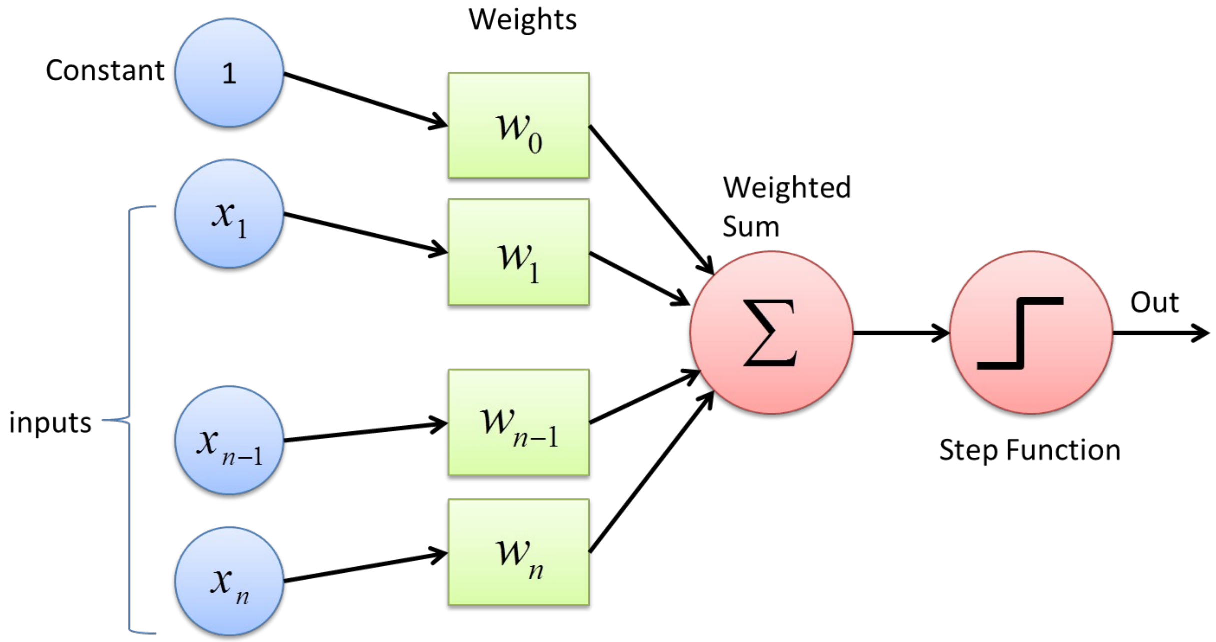
- Want to build a "binary decision tree" that splits training set on binary features for a binary class
- ID3 (Quinlan) idea:
  - Greedily pick a feature that splits the training set "as well as possible" into positive and negative subsets.
  - For each subset, recurse: pick a remaining feature to try to improve the split
  - Stop when the current subset is (almost) all one class

Information Gain

- Select next feature *f* in tree to maximize information gain
  - Recall
    - U(S) = sum[x in 0, 1] -pr(x in S) log pr(x in S)
    - Where pr(x in S) = |S[c=x]| / |S|
  - Now compute information gain *Δu* for each feature *f*
    - S+ = S[f=1]
    - S- = S[f=0]
    - Δu = u(S) - (|S+|/|S|) u(S+) - (|S-|/|S|) u(S-)
- Avoid overfitting (gain is probably just training set anomaly)
- Greedy is not optimal: mild independence assumption

Perceptrons

- "Artificial Neuron" (Papert *et al*): basis of neural nets
- Handles continuous inputs and outputs well: (we binarize)
- Idea: predict the binary class as a thresholded weighted sum of the features
- c = sum[i] w[i] x[i] + w0 > 0



https://deepai.org/machine-learning-glossary-and-terms/perceptron

Perceptron Training

- Training consists of learning appropriate weights *w*
  - Assign some initial weights
  - Feed each training instance through the perceptron
  - Adjust the weights "toward the true classification"
    - w[i] += a (c - y) x[i]
    - w0 += a (c - y)
      where *y* is the unthresholded output. Remember that *c* and *x* are 0 or 1.
      a is the "learning rate": smaller (a < 0.1) means more reliable convergence, larger *a* can mean faster learning or divergence
  - Run all the training instances repeatedly until the average accuracy isn't getting better