Artificial Intelligence

Intro to Machine Learning

Programming Style Shifts

- Standard CS: Explicitly program computer to do something
- Early AI: Derive a problem description (state) and use general algorithms to solve it
  - Search: derive a search state, successors
  - Logic: state facts as sentences, use logical inference rules to derive consequences
  - Planning: create initial state, goal state and action schemas, use forward-chaining and backward-chaining to create plans

Learning

- Now we try something different
  - Instead of giving the agent a state description, we “characterize” the set of states
  - Agent must learn what a state is

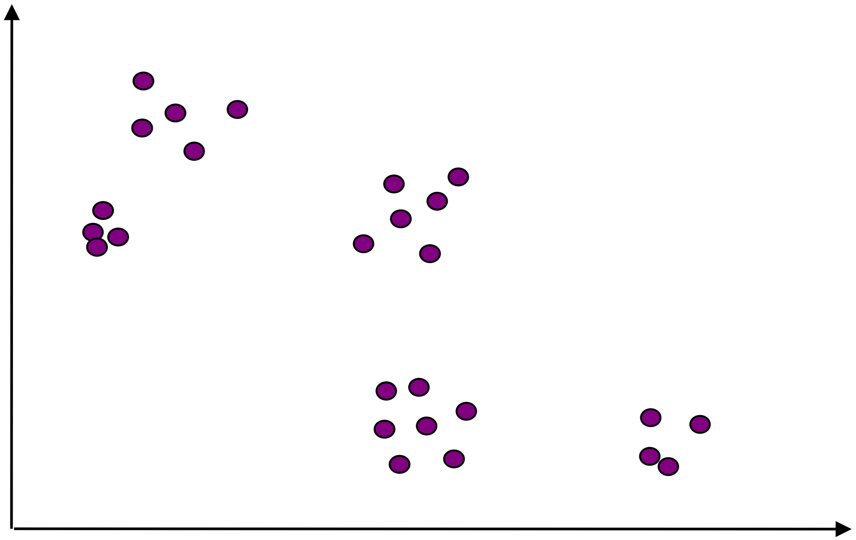
What is “Learning”?

- Agent <u>improves performance with experience</u>
  - Discover “relationships” between input and state/output
    - What features are best in mapping input to output?
    - Need to recognize what’s important and what is not
  - Discover properties of the environment

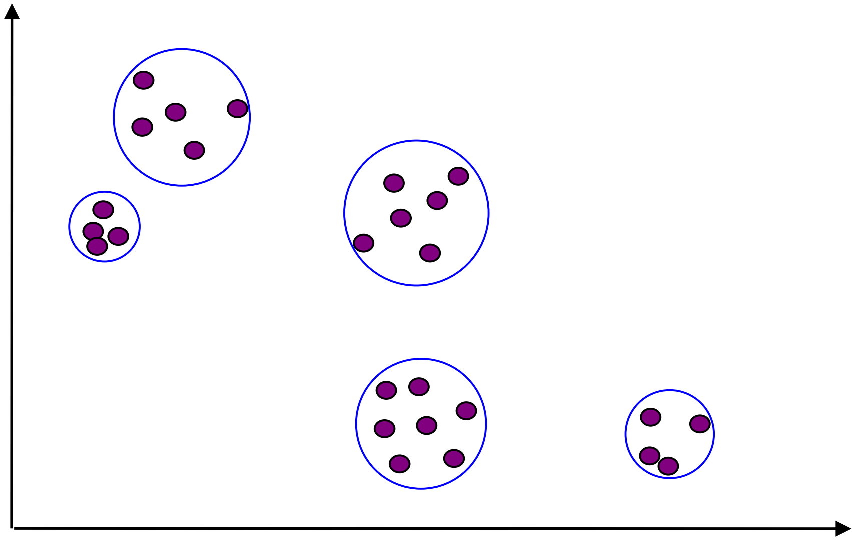
What can be Learned?

- Clusters
  - Find useful groupings of data (e.g. height/weight)
- Classifications
  - Identify hand-written digits
  - Filter mail into spam/not-spam
  - *Detect* a face in a pic
- Actions
  - Robot balances upright on two legs
  - Autopilot flies level
  - Vehicle stays in lane
  - *Locate* a face in a pic

Cluster/Grouping Learning Problems



Cluster/Grouping Learning Problems



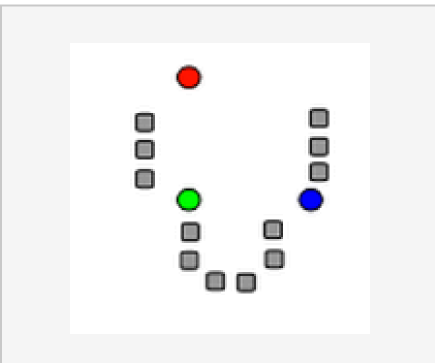
How many "groups" and
who belongs to which group?

“Unsupervised” Learning

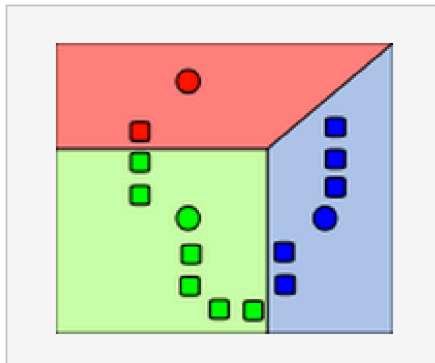
K-Means
(“Grouping/Clustering”)

- One of the <u>simplest</u> clustering algorithms, yet widely employed
- **<u>Given</u>** initial set of *K* centroids/means (generally obtained through initialization with random data points or locations):
  - Assign each point to closest (generally Euclidean) centroid
  - Recompute centroid locations based on current assignments
  - Repeat until convergence or maximum number of iterations
- Works well under constrained conditions
  - Seeding (initial centroids), cluster shapes

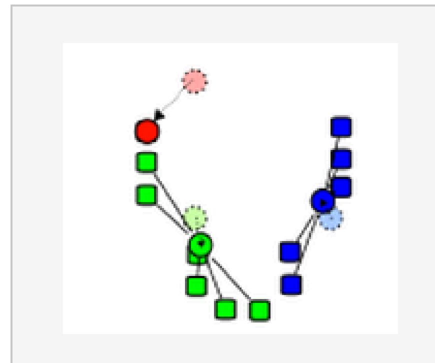
Demonstration of the standard algorithm



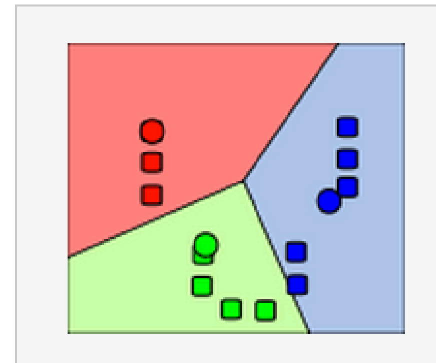
1) *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

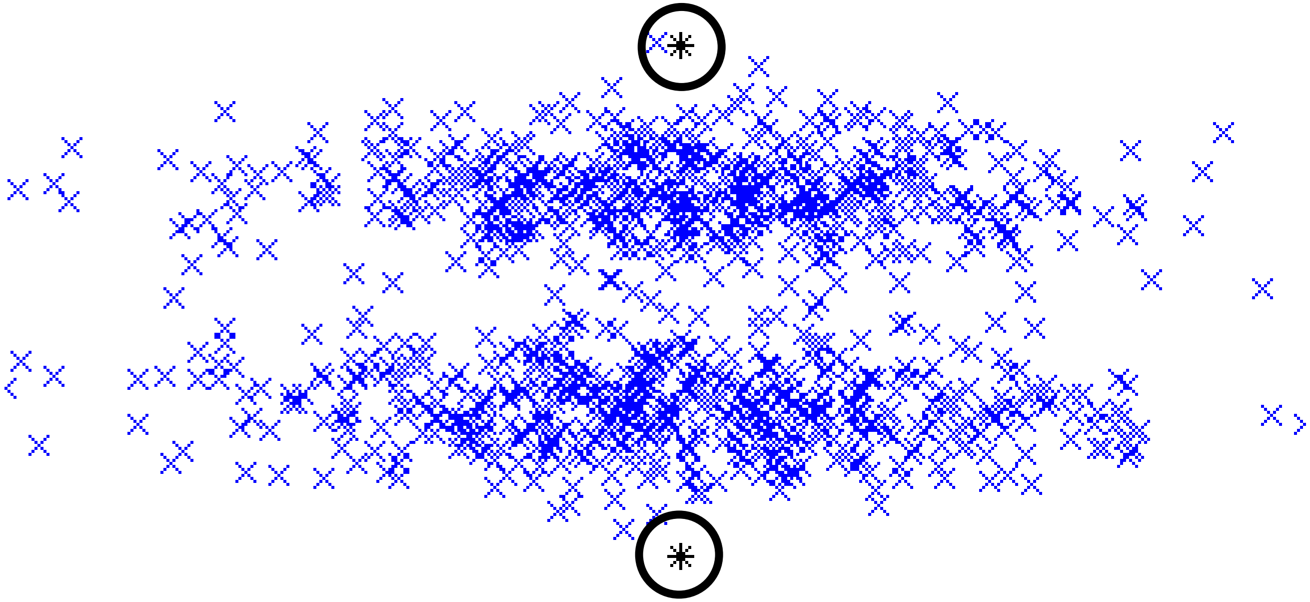


3) The centroid of each of the *k* clusters becomes the new mean.

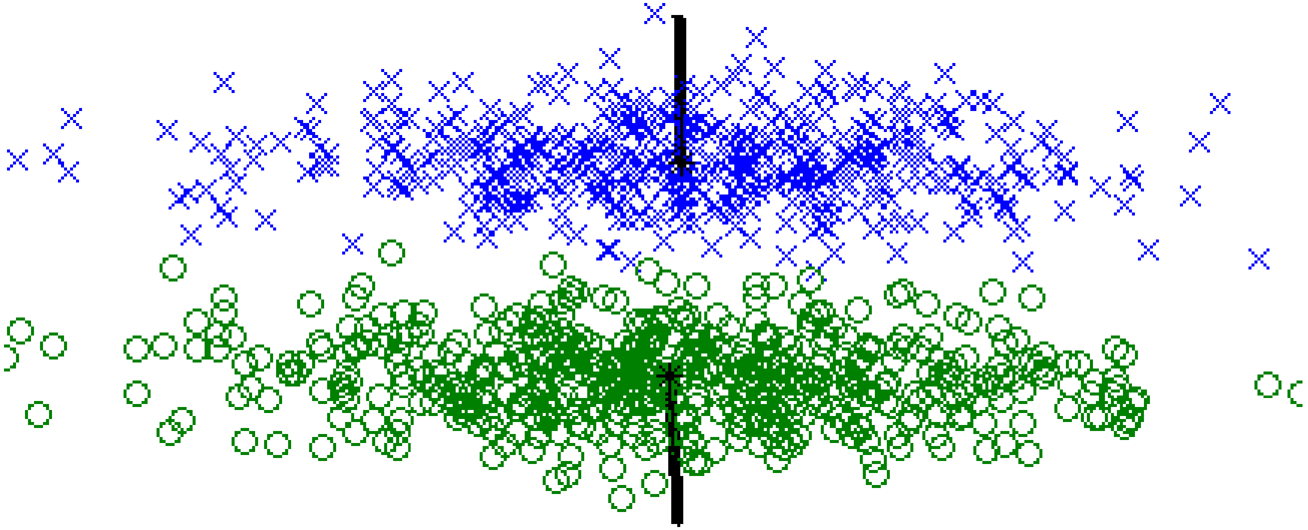


4) Steps 2 and 3 are repeated until convergence has been reached.

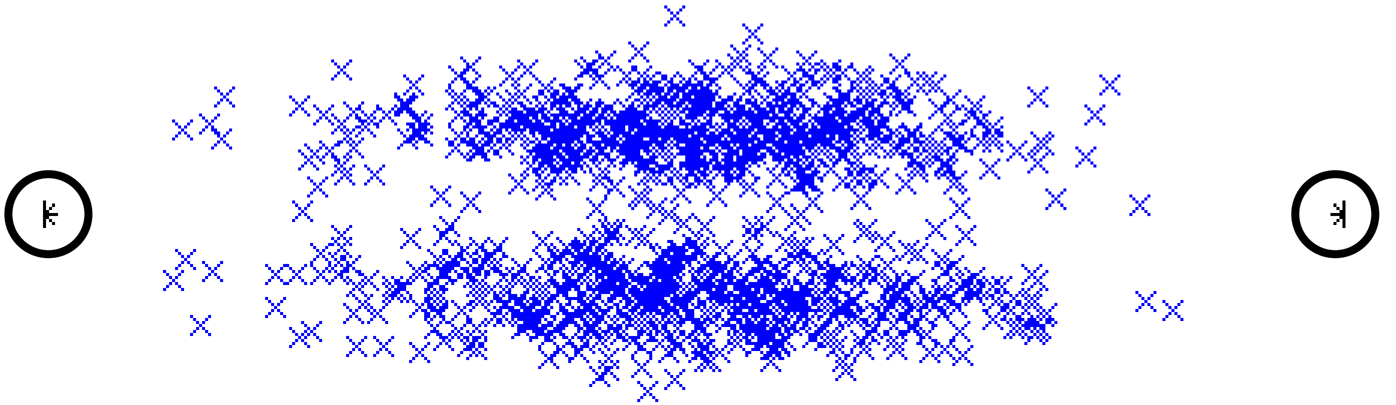
K-Means – Appropriate Seeding



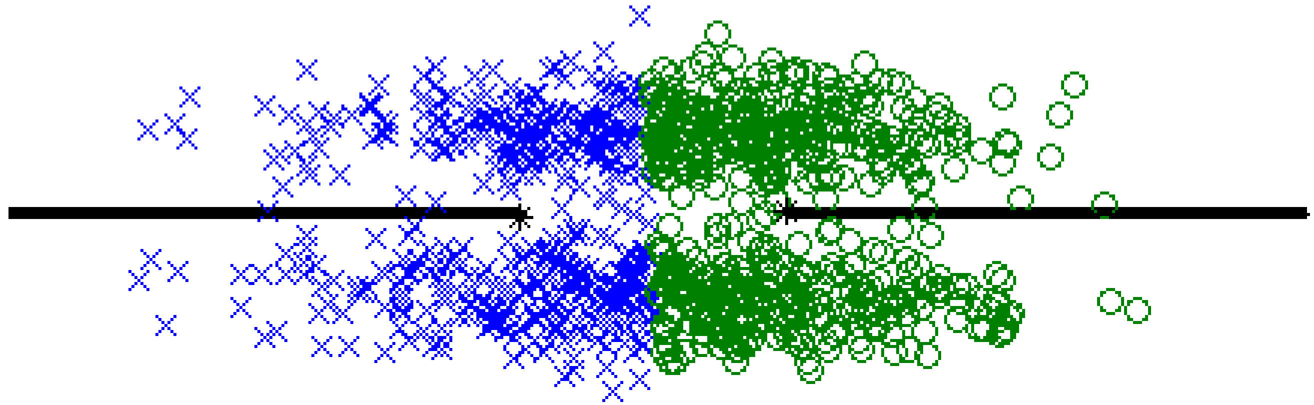
K-Means – Appropriate Seeding



K-Means – Inappropriate Seeding

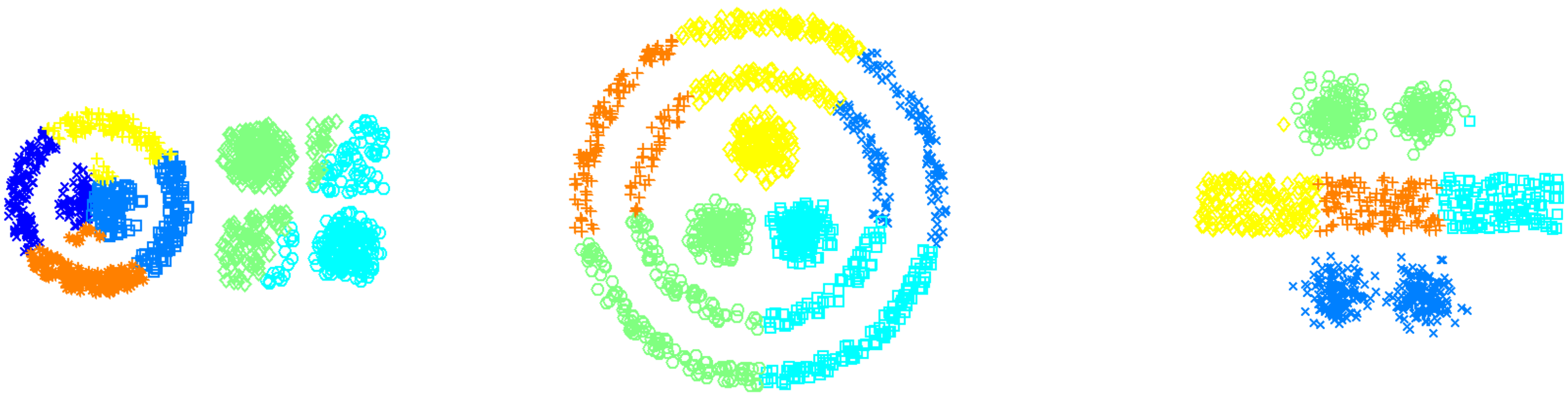


K-Means – Inappropriate Seeding



K-Means – Comments

- Results dependent on initial conditions
  - Often run multiple times and keep clustering minimizing sum of squared distances (points to centroids)
- **Need to know number of clusters *K a priori***
- Does not always perform well

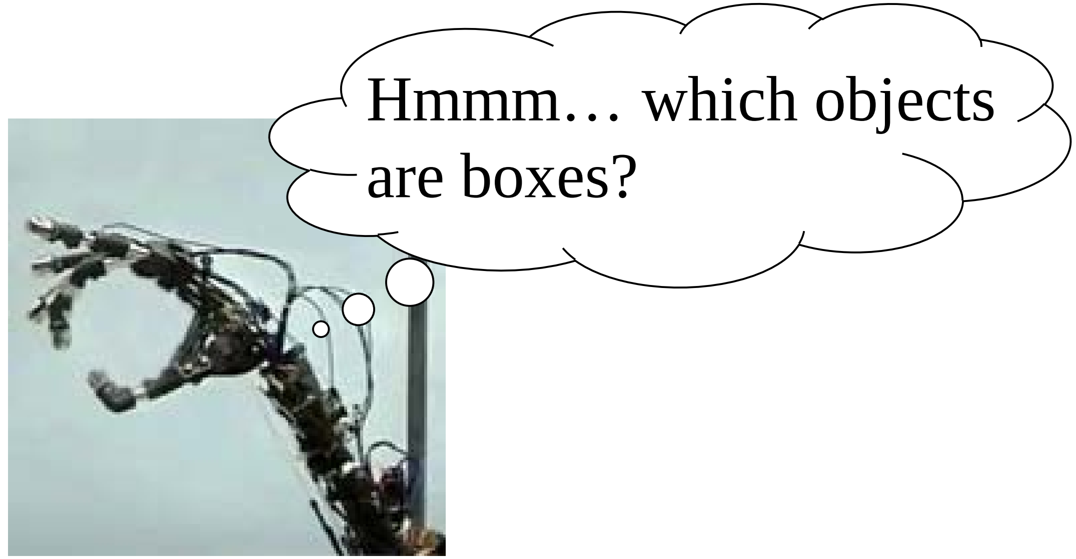


K-Means – Uses

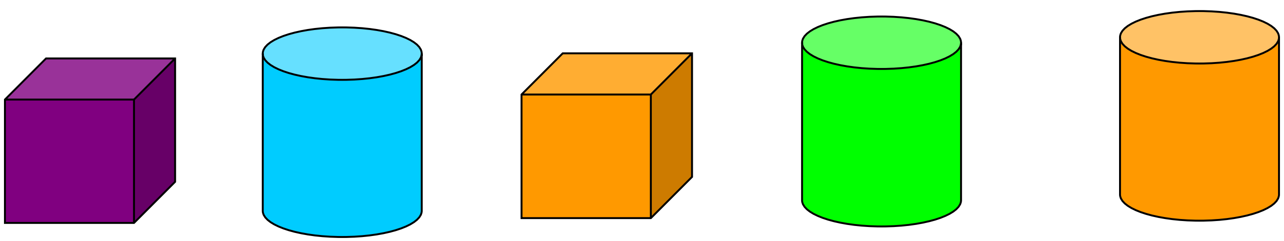
- Works well when clusters are compact and well-separated
- Often used to compress data into “prototypes” or “codewords”
  - Set K to be sufficiently large

“Supervised” Learning

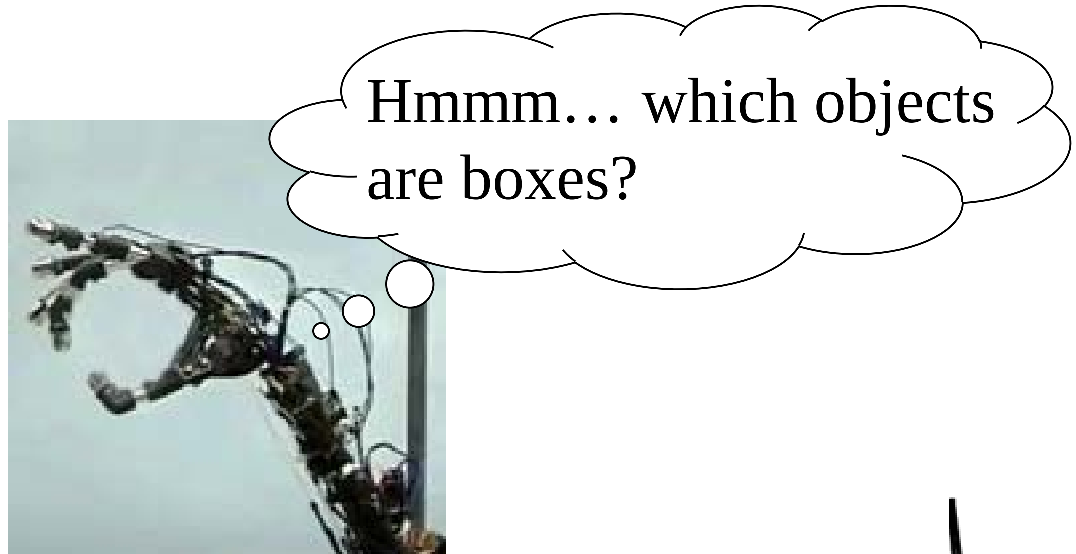
Learning



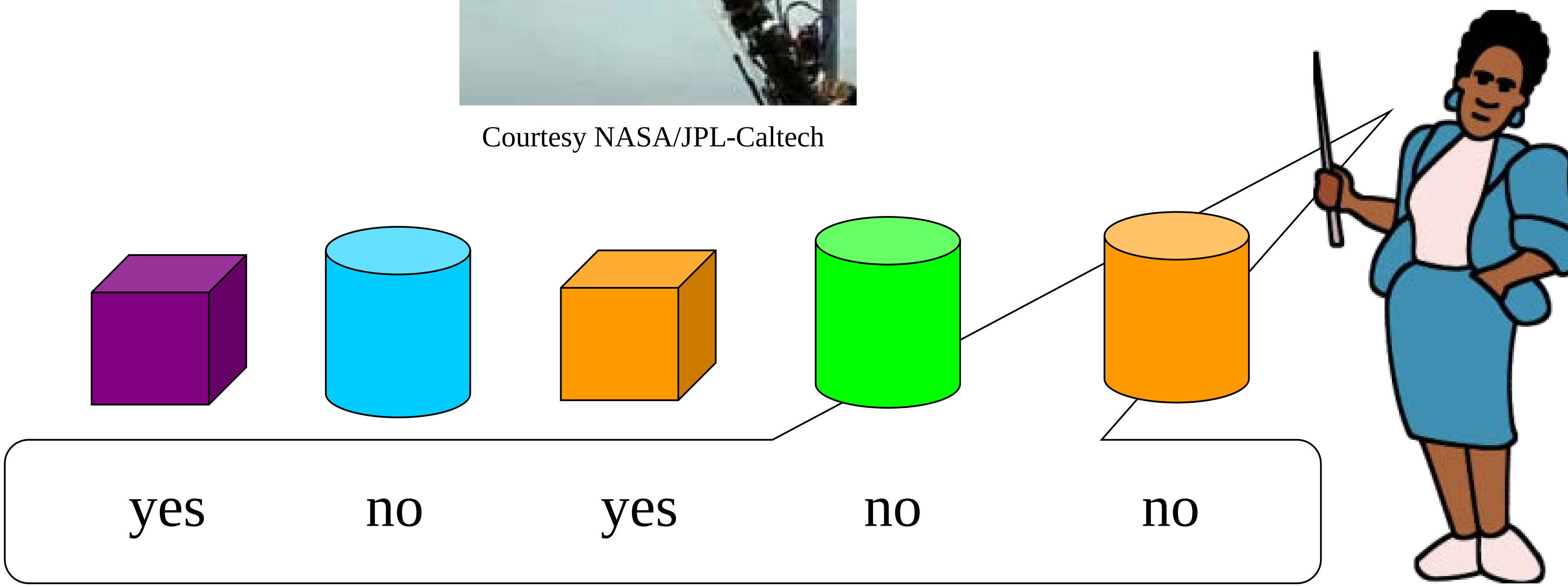
Courtesy NASA/JPL-Caltech



“Supervised” Learning



Courtesy NASA/JPL-Caltech



Supervised Learning

- Given: training data
  - Set of data <u>with corresponding class labels</u>
  - Needs to be representative of entire dataset
- Objective: build a classifier to predict output labels (classes) of data in unseen test set
  - Need to infer a function that separates the data into desirable classes
  - No single algorithm works best on all datasets
  - May need to tune algorithm parameters
  - Feature representation is important

Supervised Learning Process

- Split data into training and testing sets
- Determine features to employ
- Select a classifier
- Train the classifier using the training set
- Classify the test set
- Evaluate the classification results

Evaluating Supervised Learning

- Training data must be selected so as to reflect the global data pool
- Testing on <u>unseen data</u> is crucial to prevent *overfitting* to the training data
  - Unintended correlations between input and output
    - e.g., photos with tanks taken on sunny days
  - Correlations specific to the set of training data
    - e.g., language processing trained on Wall Street Journal or CNN transcripts may not work well for spoken conversation

Training Classifiers

- How to tune algorithm parameters?
  - "Validation"
    - Train classifier on a subset of the training data
    - Test the classifier on the remaining training data
      - Called the validation set
    - Tune the classifier to minimize the error on the validation set

Training Classifiers (continued)

- How to tune? (continued)
  - *m*-Fold Cross-Validation
    - Set classifier options
      - e.g., number of parameters, model form, training time, input features, etc.
    - Estimate **generalized** classifier performance
      - Randomly divide training set into *m* disjoint sets of equal size
      - Train using (*m*-1) subsets and validate on the remaining subset
      - Repeat *m* times, using different validation set each time
      - Average results
    - Repeat entire process for different classifier options and choose the options which maximize the average results

Evaluation

- Accuracy = $\dfrac{\text{Number of correct classifications}}{\text{Number of classifications}}$

- Consider the case where we are trying to detect instances of class X within a dataset containing instances from X and Y
  - True Positive (TP) – Correctly classifying an instance of X as X
  - False Positive (FP) – Incorrectly classifying an instance of Y as X
    - False alarm or Type I error
  - True Negative (TN) – Correctly classifying an instance of Y as Y
  - False Negative (FN) – Incorrectly classifying an instance of X as Y
    - Misdetection or Type II error

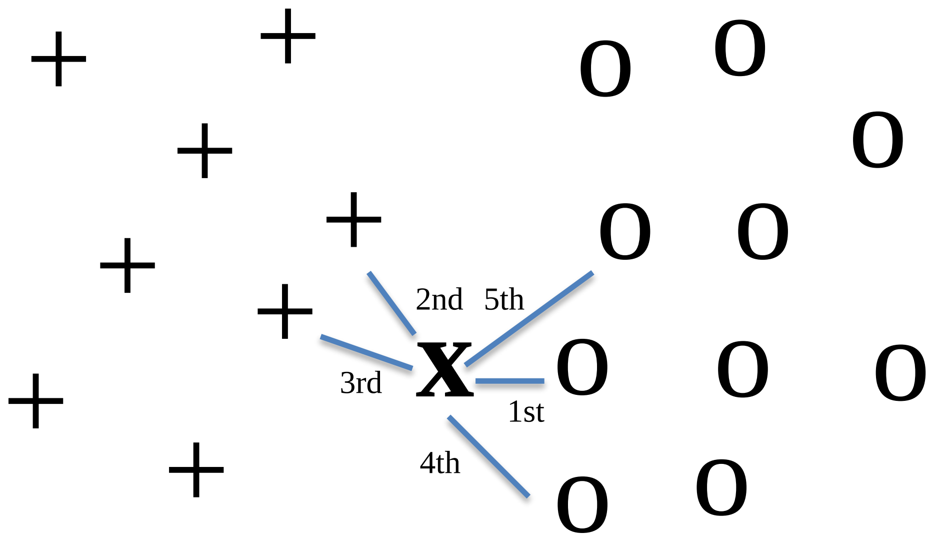
Evaluation

- $\text{Precision} = \dfrac{\text{Number of correctly detected events}}{\text{Number of detected events}} = \dfrac{\text{TP}}{\text{TP} + \text{FP}}$
  - For low values, the algorithm is saying "yes" in many cases where it shouldn't (false positives)
  - It is not being as *precise* as it should be in saying "yes"
- $\text{Recall} = \dfrac{\text{Number of correctly detected events}}{\text{True number of events}} = \dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  - For low values, the algorithm is not saying "yes" everywhere it should (false negatives)
  - It is not *recalling* every "yes" it should
- $F_\beta - \text{Measure} = \left(1 + \beta^2\right) \cdot \dfrac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$
  - Common to use $\beta = 1 \rightarrow$ Harmonic mean between precision and recall

k-NN

k-Nearest Neighbor

- One of the <u>simplest</u> classification strategies
- Algorithm:
  - Compute <u>distance</u> from test sample to labeled training samples
  - Assign test sample the label <u>most common</u> across the first *k* nearest neighbors from the training data
    - *k* typically small and odd numbered (no ties)



K=1 yields X is class o

K=3 yields X is class +

K=5 yields X is class o

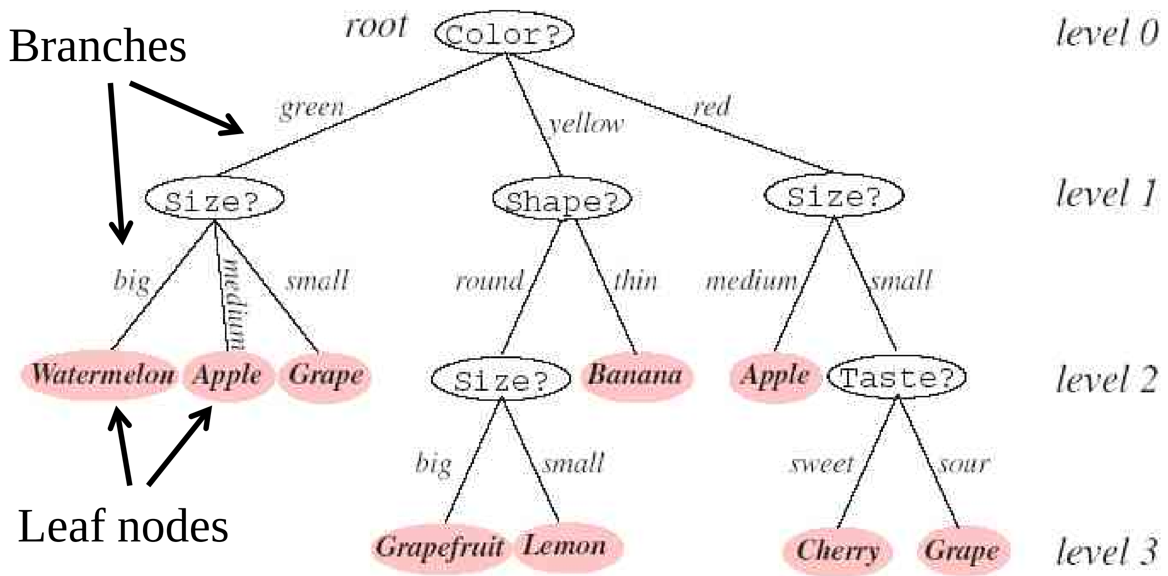
Decision Tree

Decision Trees

- Input: Feature vectors
- Output: Classification of input vector
- Learns by subdividing the data into clusters with same properties
- Good at determining which features are good discriminators

Decision Tree

- Classify pattern through sequence of questions
- Easy to interpret

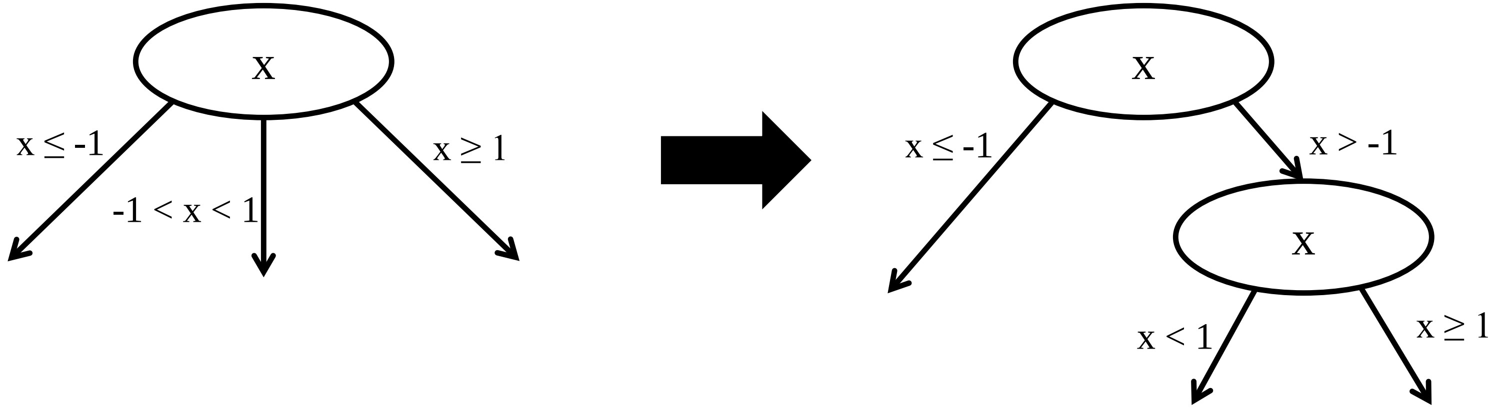


CART

- **Classification and Regression Trees** (CART)
  - General framework for creating decision trees
- Common questions:
  1. How many splits at each node?
  2. Which property should be tested at node?
  3. When should the tree stop?
  4. Can "large" trees be pruned (to make smaller)?

1) How Many Splits?

- Every <u>non-binary</u> decision can be represented as <u>combination of binary</u> decisions



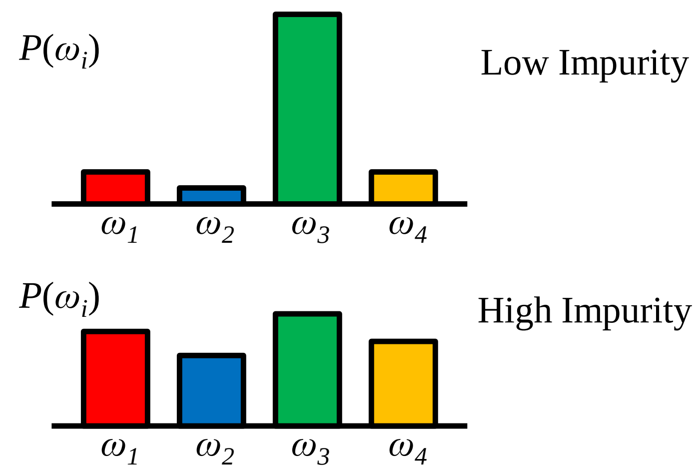
2) Which Property to Test?

- Prefer decisions that lead to simplest tree (Occam's Razor)
  - Want property to split data into "purest" groups possible
  - Use <u>impurity</u> measures

Node N

Proportion of patterns at node N that belong to class ω_j

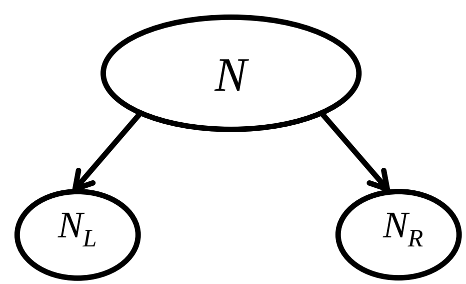
Entropy Impurity $i(N) = -\sum_j P(\omega_j)\log_2 P(\omega_j)$



- Choose decision at node N that <u>decreases</u> impurity the most
  - Maximize $\Delta i(N) = i(N) - \overbrace{\left[P_L \cdot i(N_L) + (1 - P_L) \cdot i(N_R)\right]}^{\text{Want} \ll i(N)}$

Proportion of patterns that go out to node N_L

Impurity of patterns at node N_L



3) When to Stop Splitting?

- Four different techniques:
  1. Continue splitting until training error on validation data is minimized
  2. Below threshold value in impurity reduction
  3. Minimize cost function balancing tree size and impurity
  4. Test statistical significance of impurity reduction

4) How to Prune Large Trees?

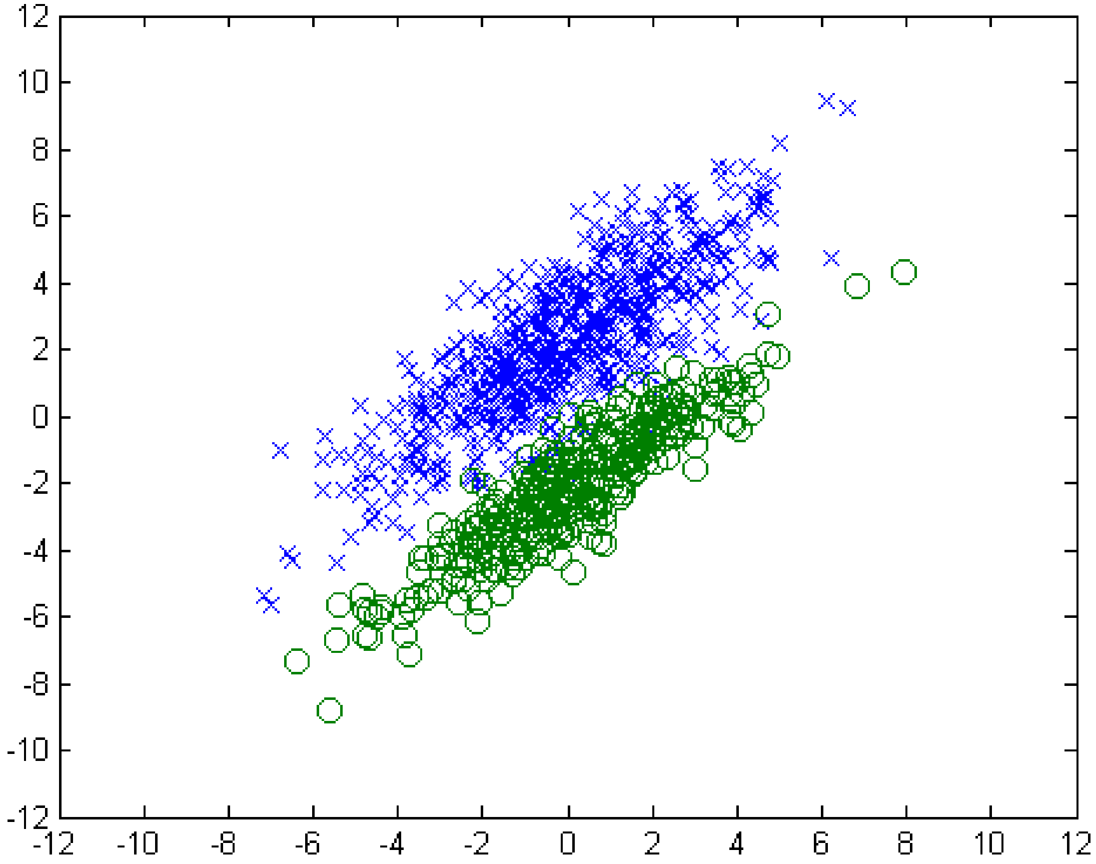
- Methods to determine when to stop splitting may declare a node a leaf too early
- Alternative: grow tree out entirely (each leaf perfectly pure) and then prune
- Pruning:
  - Work bottom-up
  - Compute the increase in impurity if two child nodes linked to common parent node are eliminated
  - Merge if increase is negligible

Lastly Assign Categories to Leaf Nodes

- Simplest approach is to take majority vote of class labels at leaf node
  - Ideally there will be one dominant class
- Potential options when tie occurs:
  - Random assignment
  - Take into account priors
  - Take into account classification risks
    - Cost of misdetections or false alarms of categories

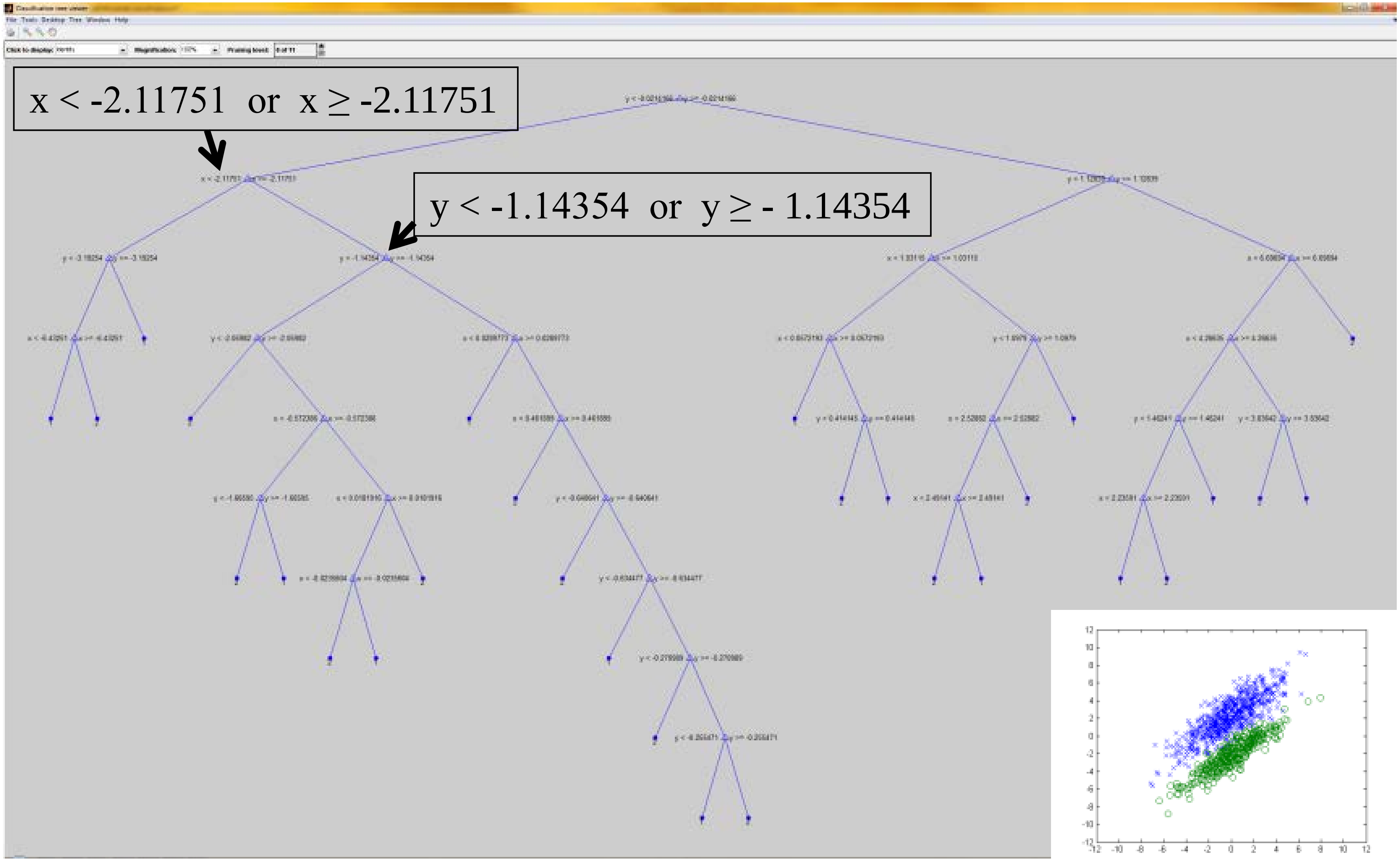
Example

TASK: Build decision tree to represent these two data clusters/classes.
(Then can push new "unlabeled" point into tree to determine its class.)



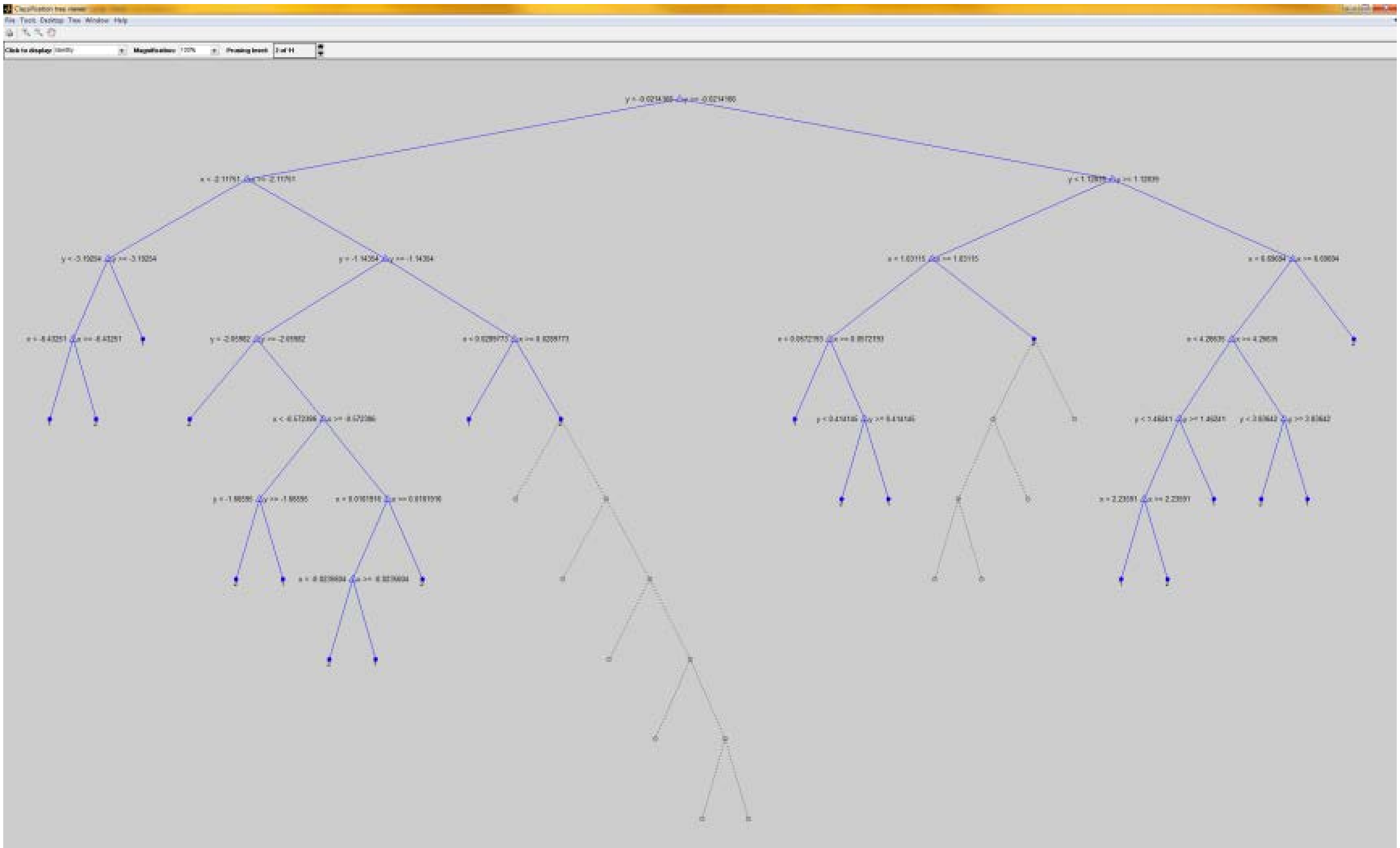
Example – Decision Tree (Matlab)

Mostly Full Tree (nodes must have at least 10 observations to be split)



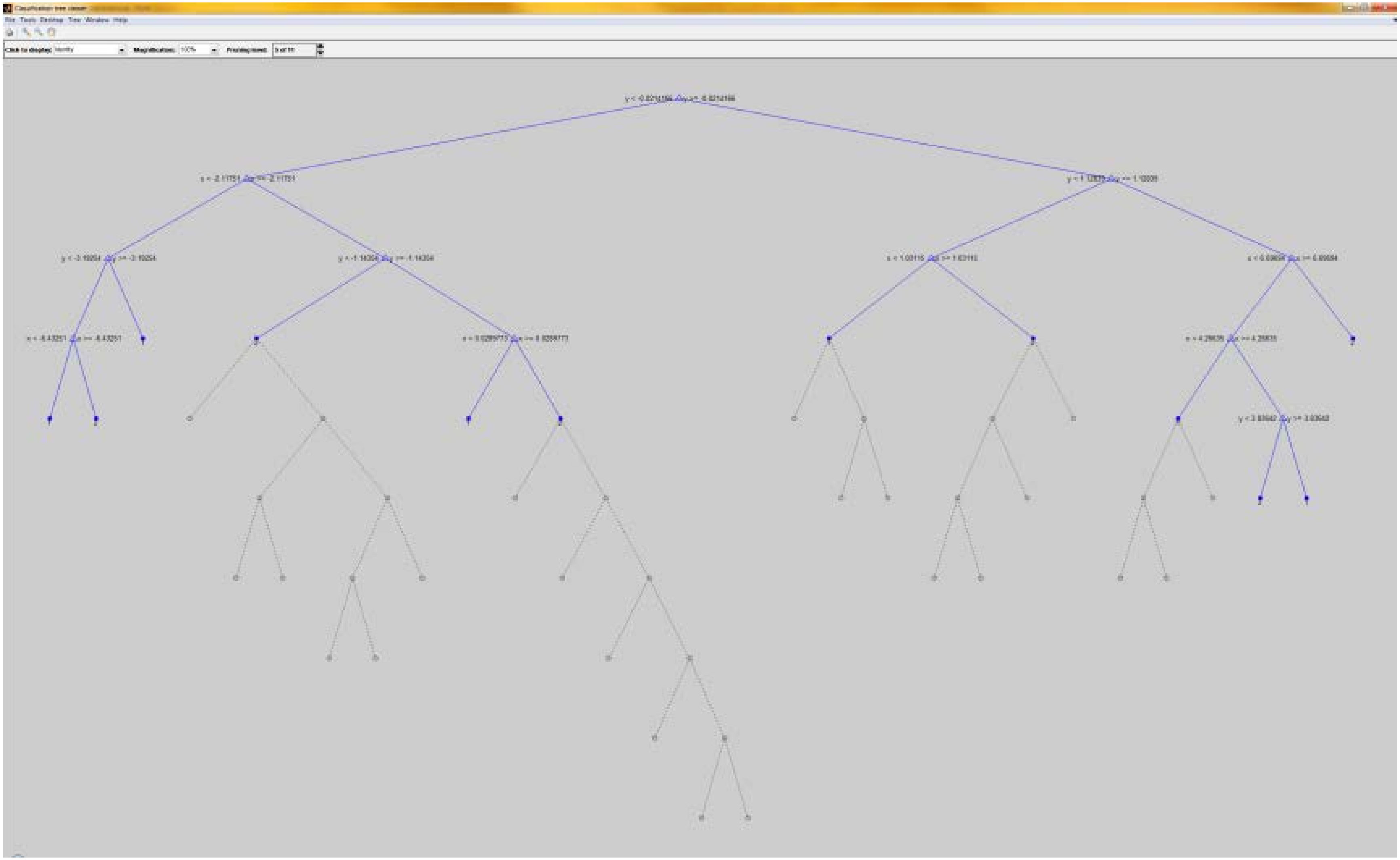
Example – Decision Tree

Slightly Pruned Tree



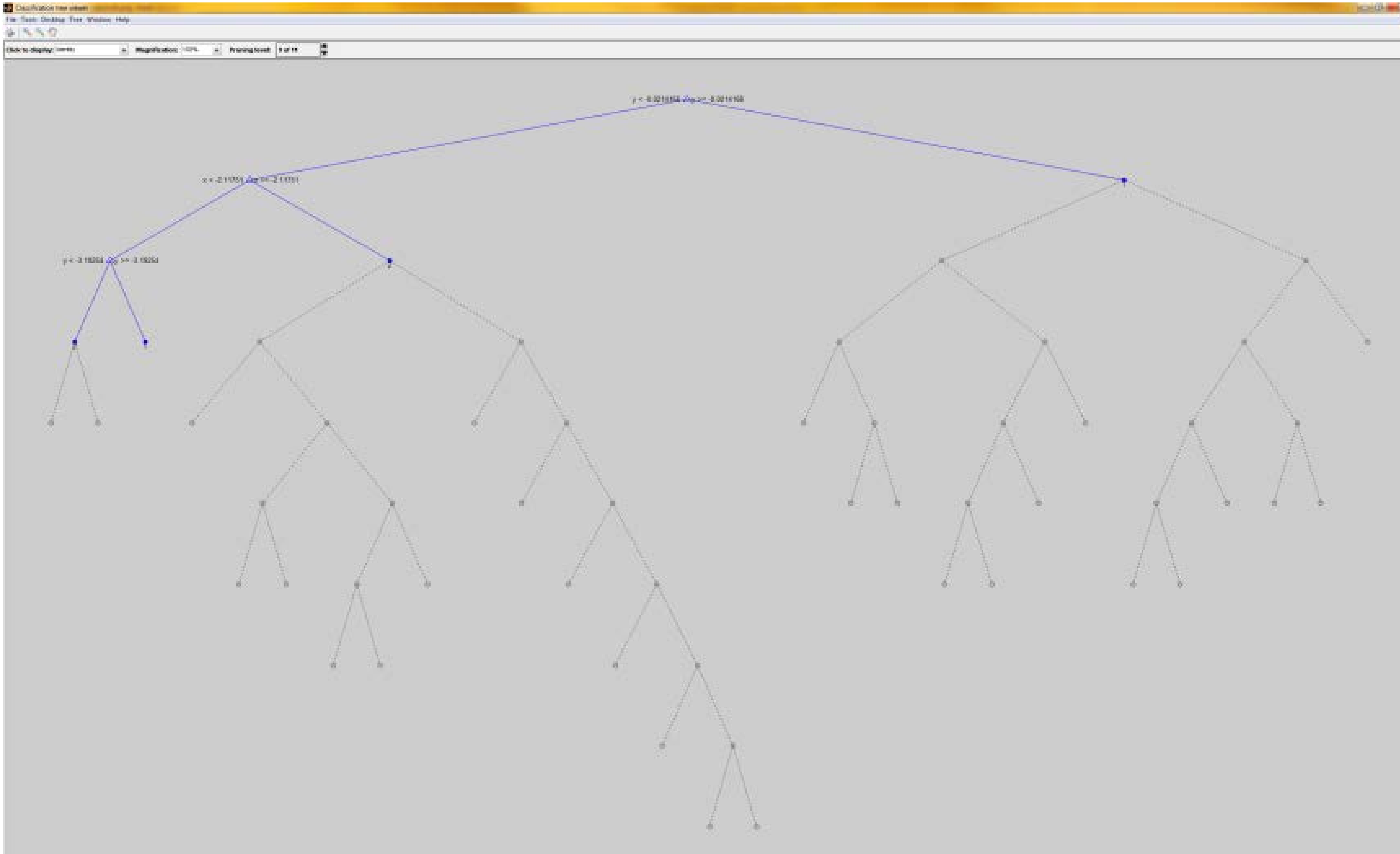
Example – Decision Tree

Moderately Pruned Tree



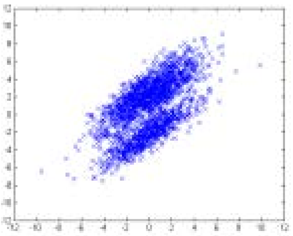
Example – Decision Tree

Heavily Pruned Tree



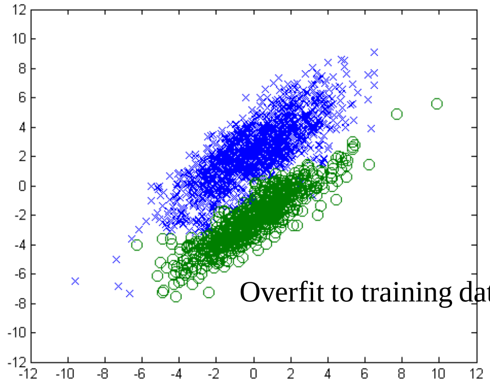
Example – Decision Tree

Unlabeled test data



Results from test data

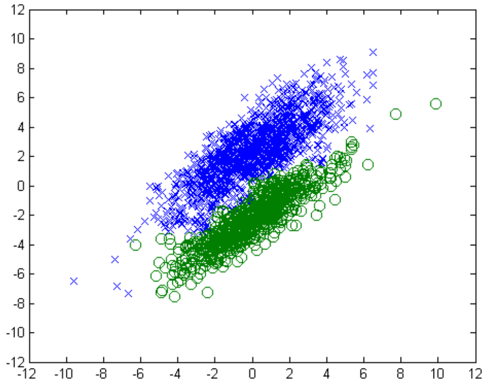
Mostly Full Tree



Overfit to training data

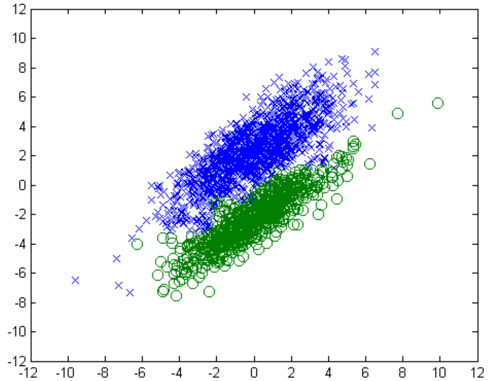
Accuracy = 0.9635

Slightly Pruned Tree



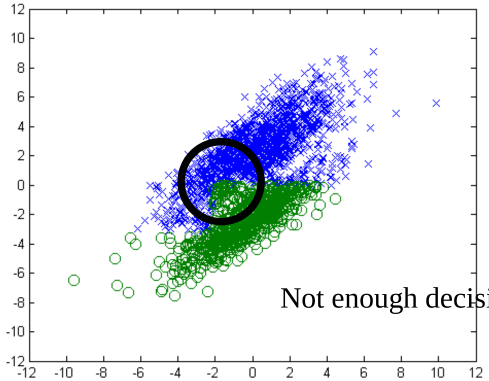
Accuracy = 0.9655

Moderately Pruned Tree



Accuracy = 0.9670

Highly Pruned Tree



Not enough decisions

Accuracy = 0.9165

Summary

- Unsupervised learning
  - K-Means
- Supervised learning
  - Training and evaluation
  - *k*-NN
    - Majority vote of nearest neighbors in training data
  - Decision Trees
    - CART