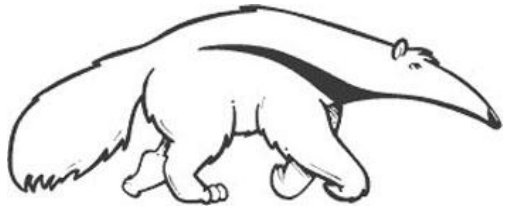
Machine Learning Classifiers: Many Diverse Ways to Learn

CS271P, Fall Quarter, 2018
Introduction to Artificial Intelligence
Prof. Richard Lathrop



Read Beforehand: R&N 18.5-12, 20.2.2

You will be expected to know

- Classifiers:
  - Decision trees
  - K-nearest neighbors
  - Perceptrons
  - Support vector Machines (SVMs), Neural Networks
  - Naïve Bayes

- Decision Boundaries for various classifiers
  - What can they represent conveniently? What not?

Review: Supervised Learning

Supervised learning: learn mapping, attributes $\rightarrow$ target
– Classification: target variable is discrete (e.g., spam email)
– Regression: target variable is real-valued (e.g., stock market)

Review: Supervised Learning

Supervised learning: learn mapping, attributes $\rightarrow$ target
– Classification: target variable is discrete (e.g., spam email)
– Regression: target variable is real-valued (e.g., stock market)

Simple illustrative learning problem

Problem:

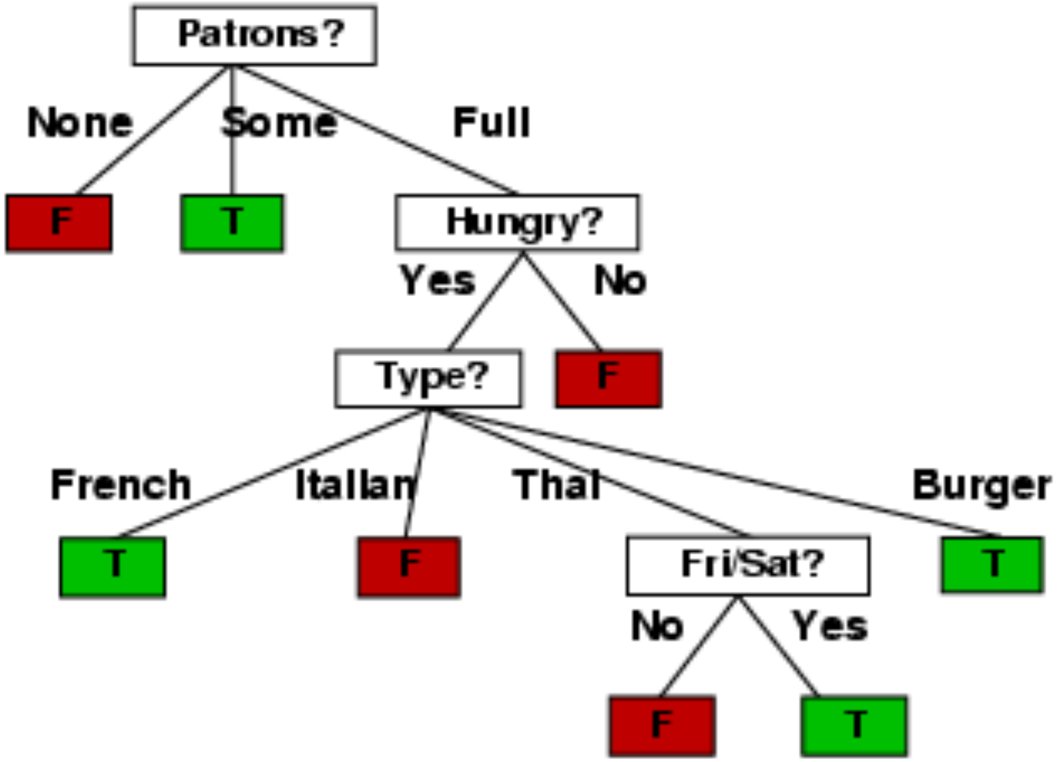
Decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Review: Training Data for Supervised Learning

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Review: Decision Tree



Review: Supervised Learning

- Let $\underline{x}$ represent the input vector of attributes
  - x_j is the value of the jth attribute, $j = 1, 2, \ldots, d$

- Let $f(\underline{x})$ represent the value of the target variable for $\underline{x}$
  - The implicit mapping from x to $f(\underline{x})$ is unknown to us
  - We just have training data pairs, $D = \{\underline{x}, f(\underline{x})\}$ available

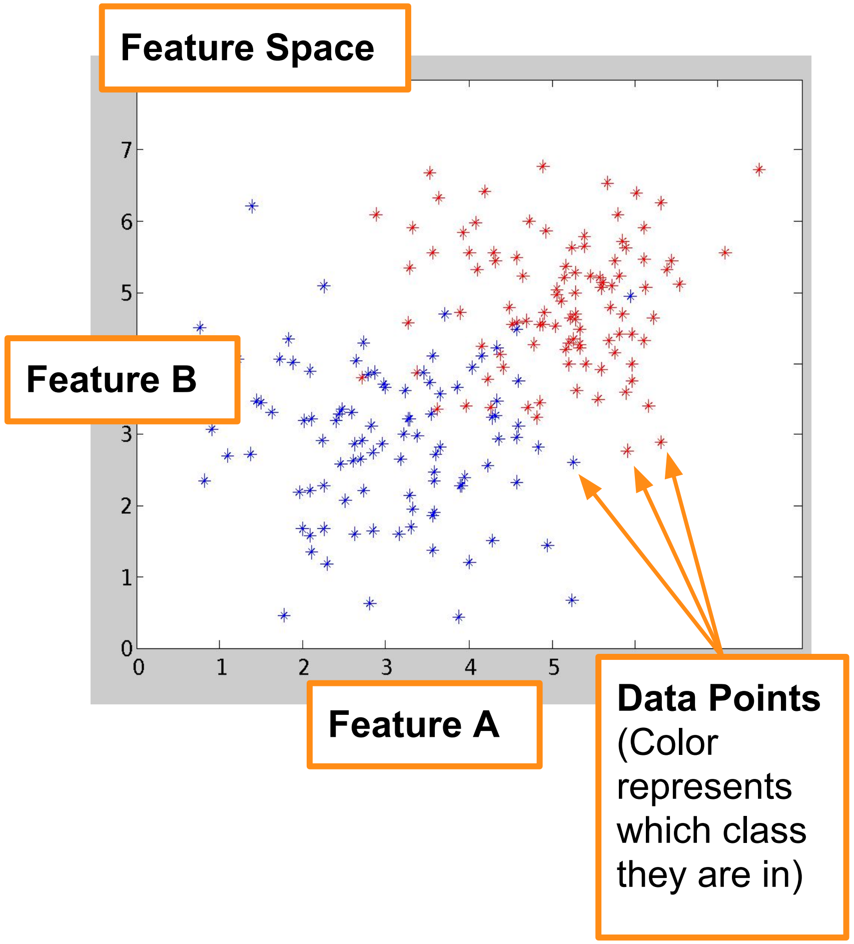
- We want to learn a mapping from $\underline{x}$ to f, i.e.,
  - $h(\underline{x}; \theta)$ should be "close" to $f(x)$ for all training data points $\underline{x}$

    θ are the parameters of the hypothesis function h( )

- Examples:
  - $h(\underline{x}; \theta) = \text{sign}(w_1 x_1 + w_2 x_2 + w_3)$
  - $h_k(\underline{x}) = (x_1 \text{ OR } x_2) \text{ AND } (x_3 \text{ OR NOT}(x_4))$

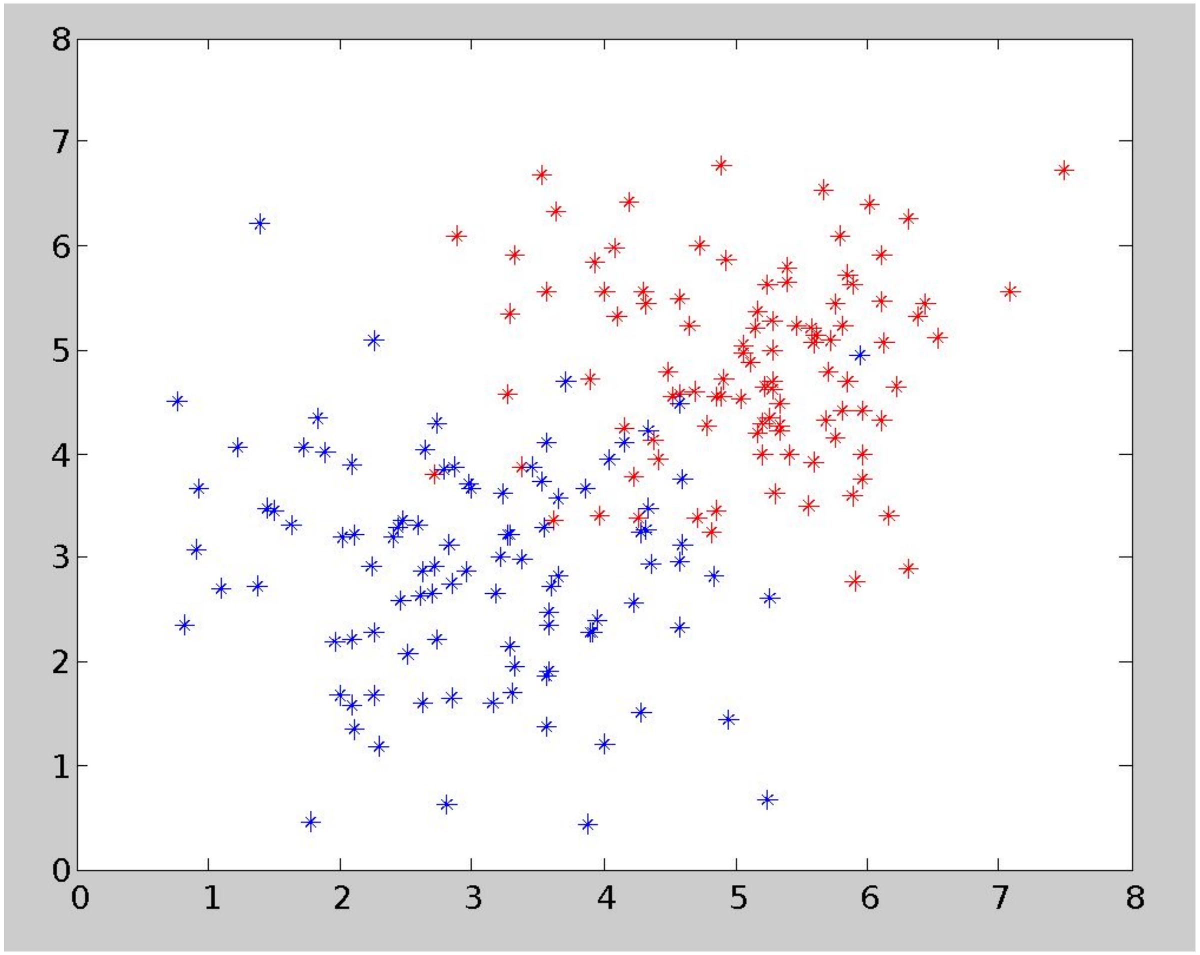
A Different View on Data Representation

- Data pairs can be plotted in "feature space"
- Each axis represents 1 feature.
  - This is a d dimensional space, where d is the number of features.
- Each data case corresponds to 1 point in the space.
  - In this figure we use color to represent their class label.

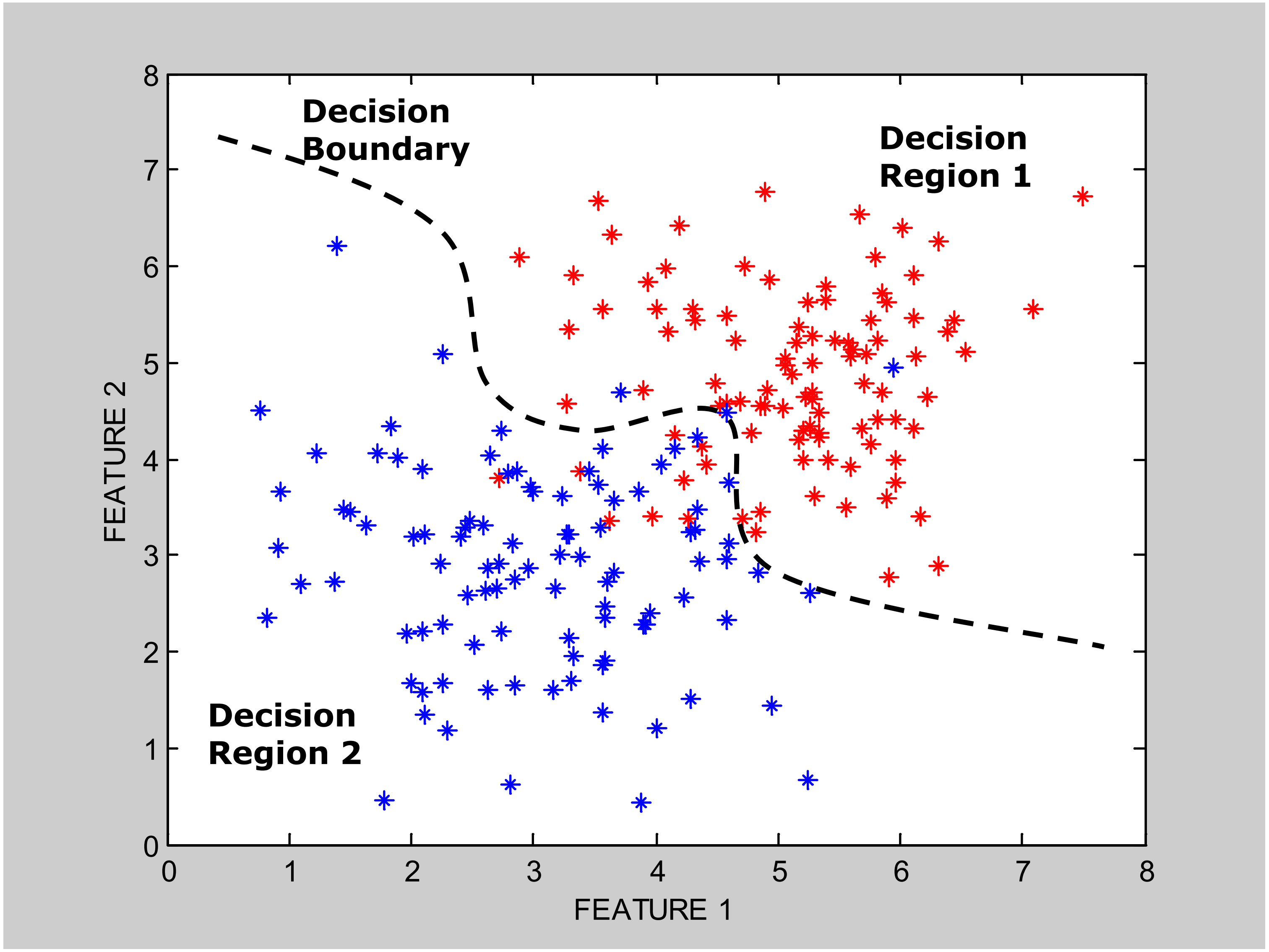


Decision Boundaries

Can we find a boundary that separates the two classes?



Decision Boundaries



Classification in Euclidean Space

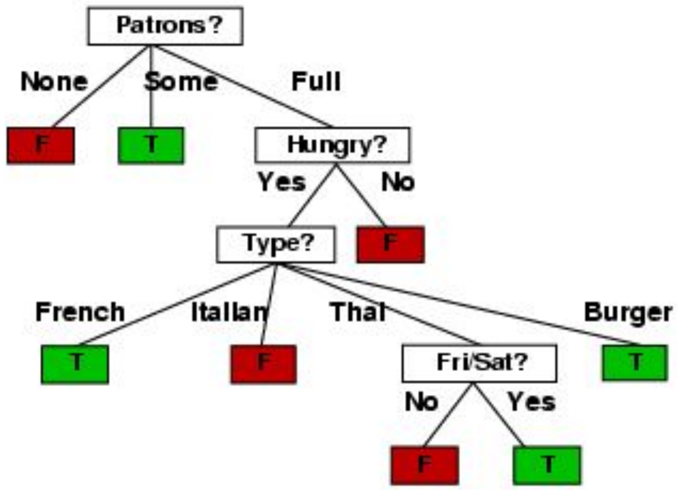
- A classifier is a partition of the feature space into disjoint decision regions
  - Each region has a label attached
  - Regions with the same label need not be contiguous
  - For a new test point, find what decision region it is in, and predict the corresponding label

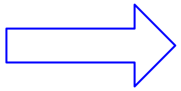
- Decision boundaries = boundaries between decision regions

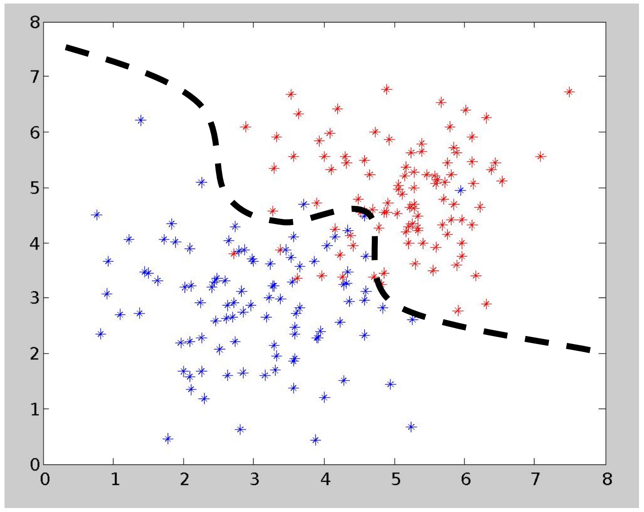
- We can characterize a classifier by the equations for its decision boundaries

- Learning a classifier $\Leftrightarrow$ searching for the decision boundaries that optimize our objective function

Can we represent a decision tree classifier in the feature space?







Example: Decision Trees

- When applied to continuous attributes, decision trees produce "axis-parallel" linear decision boundaries

- Categorical features -> values from a discrete set

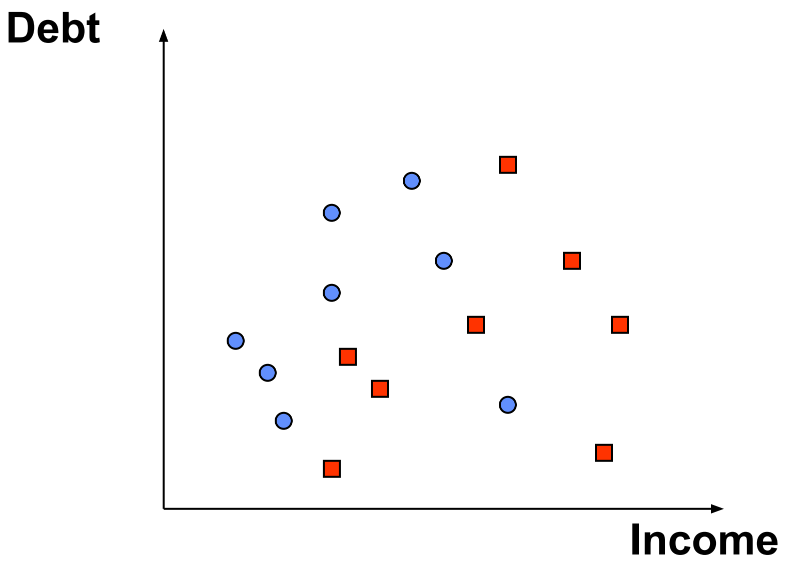
  e.g. Restaurant type (French, Italian, Thai, Burger)
  Raining outside? (Yes/No)

- Continuous features -> real values

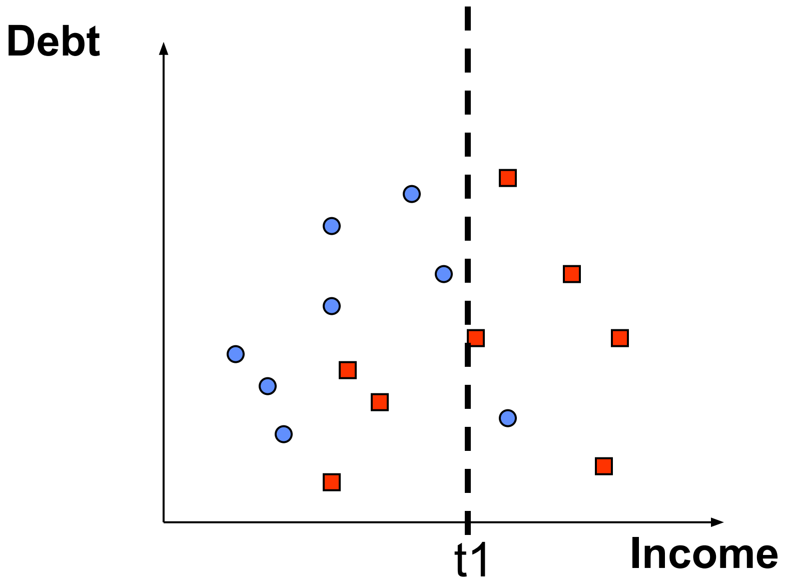
  e.g. Income

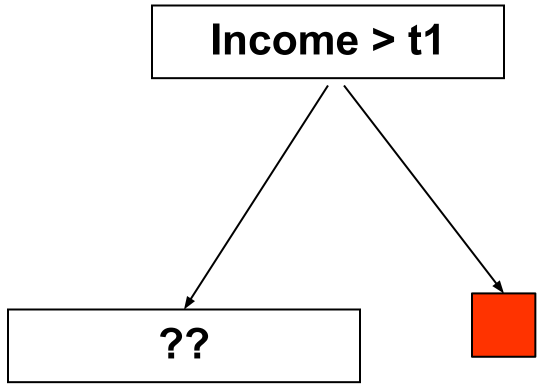
  – Each internal node is a binary threshold of the form $x_j > t$? and converts each real-valued feature into a binary one

Decision Tree Example

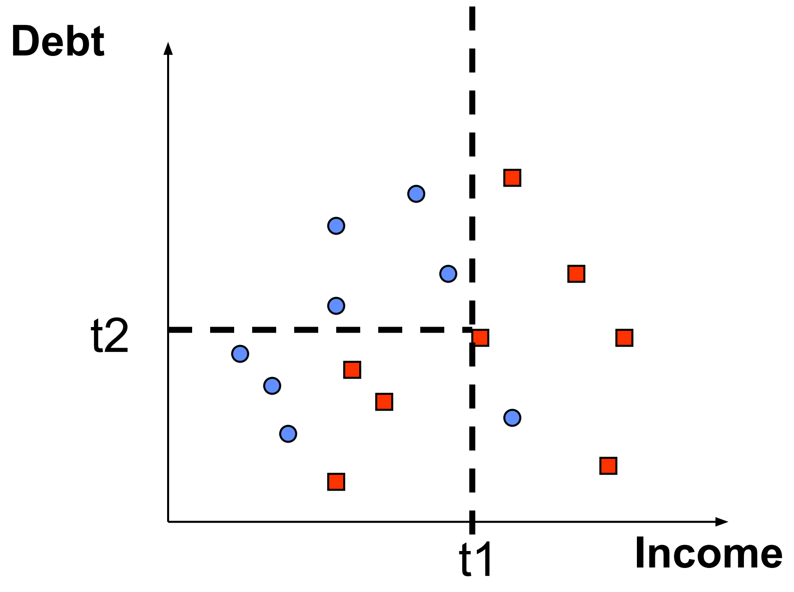


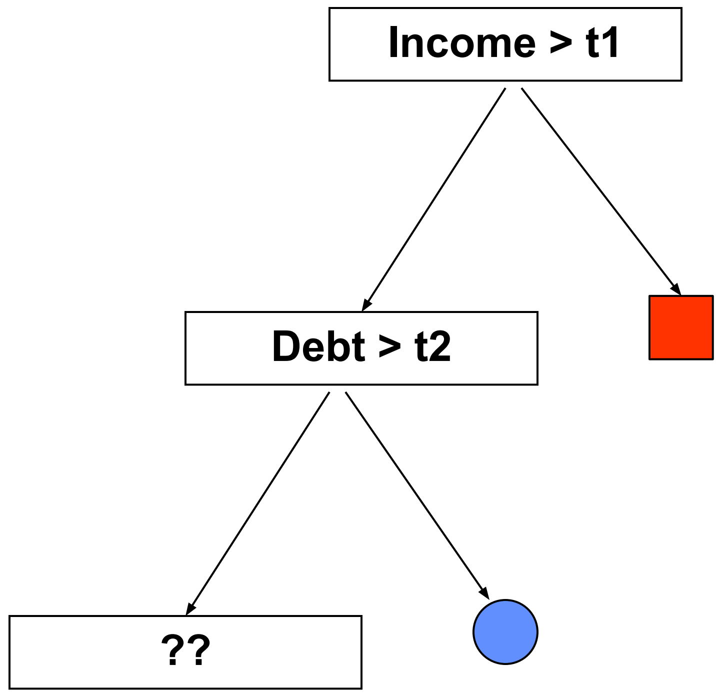
Decision Tree Example



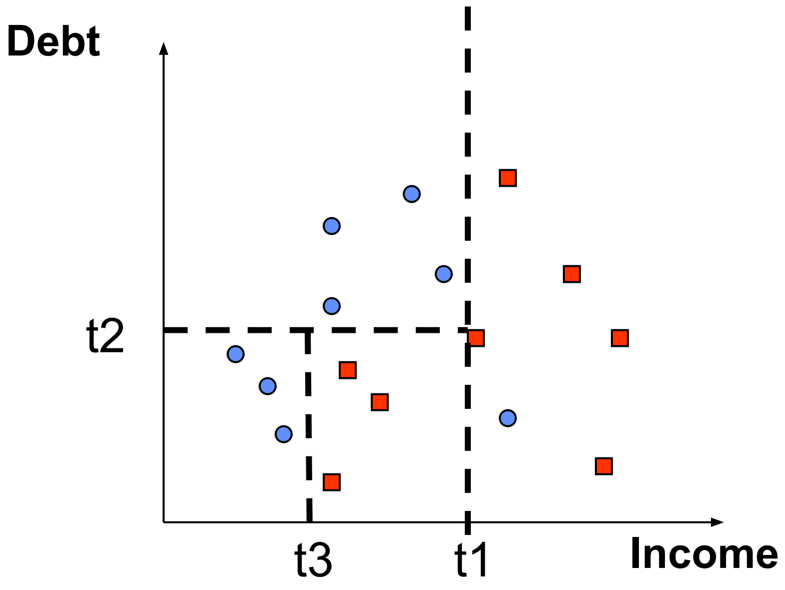


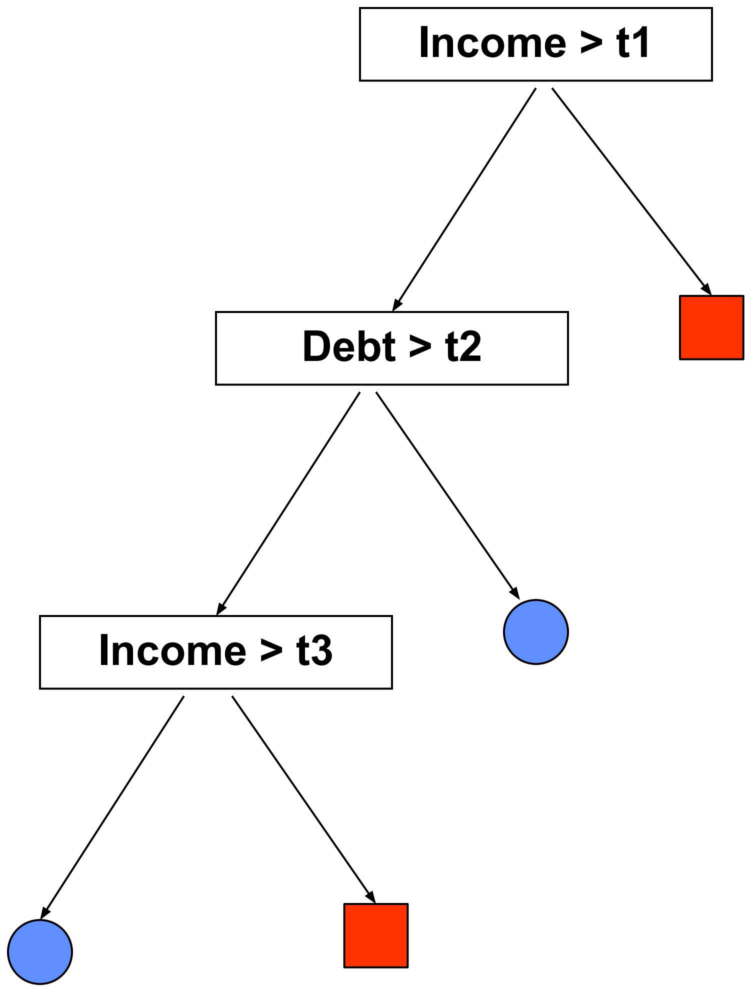
Decision Tree Example



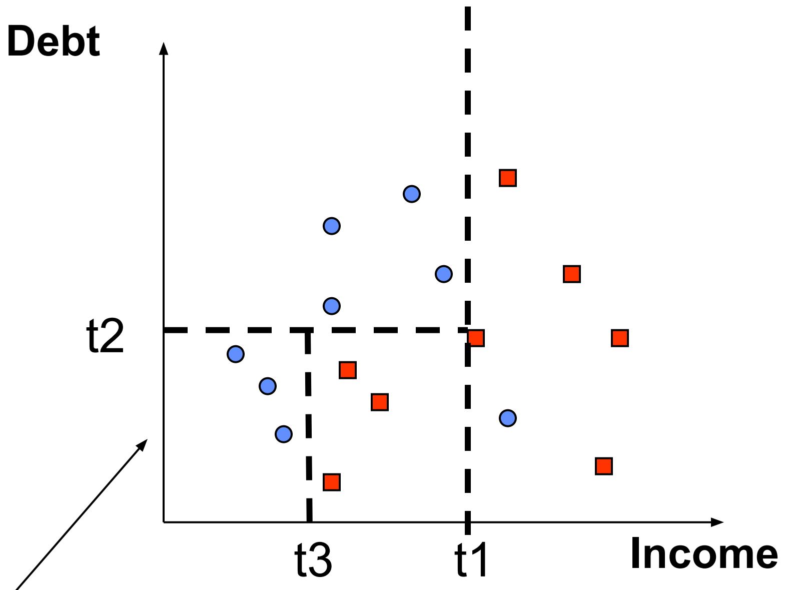


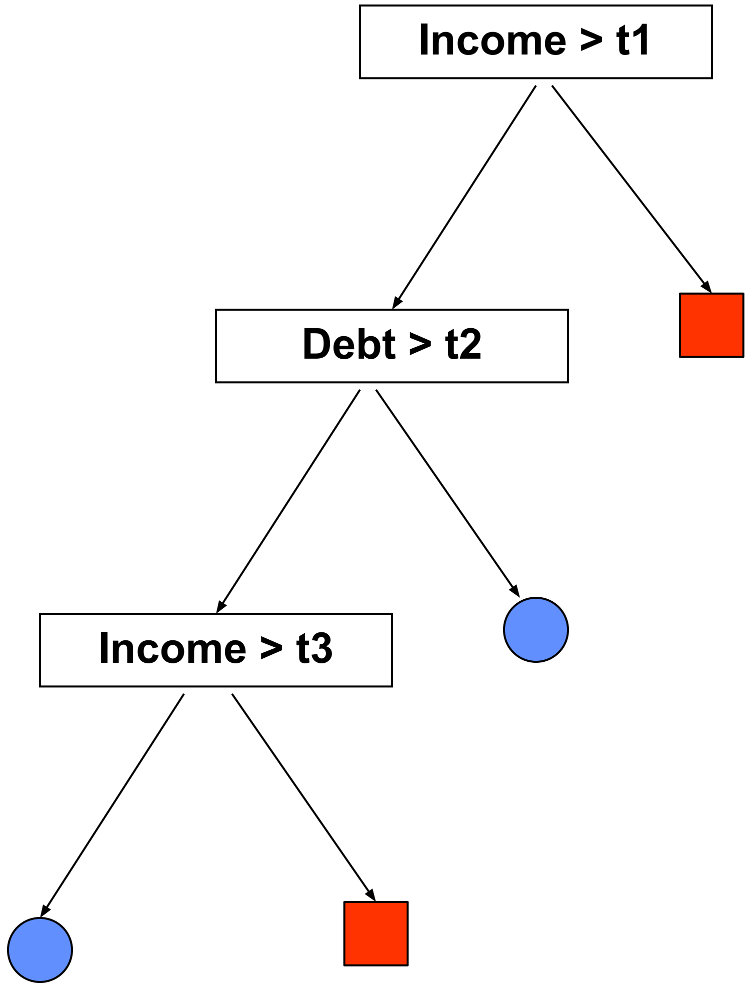
Decision Tree Example





Decision Tree Example



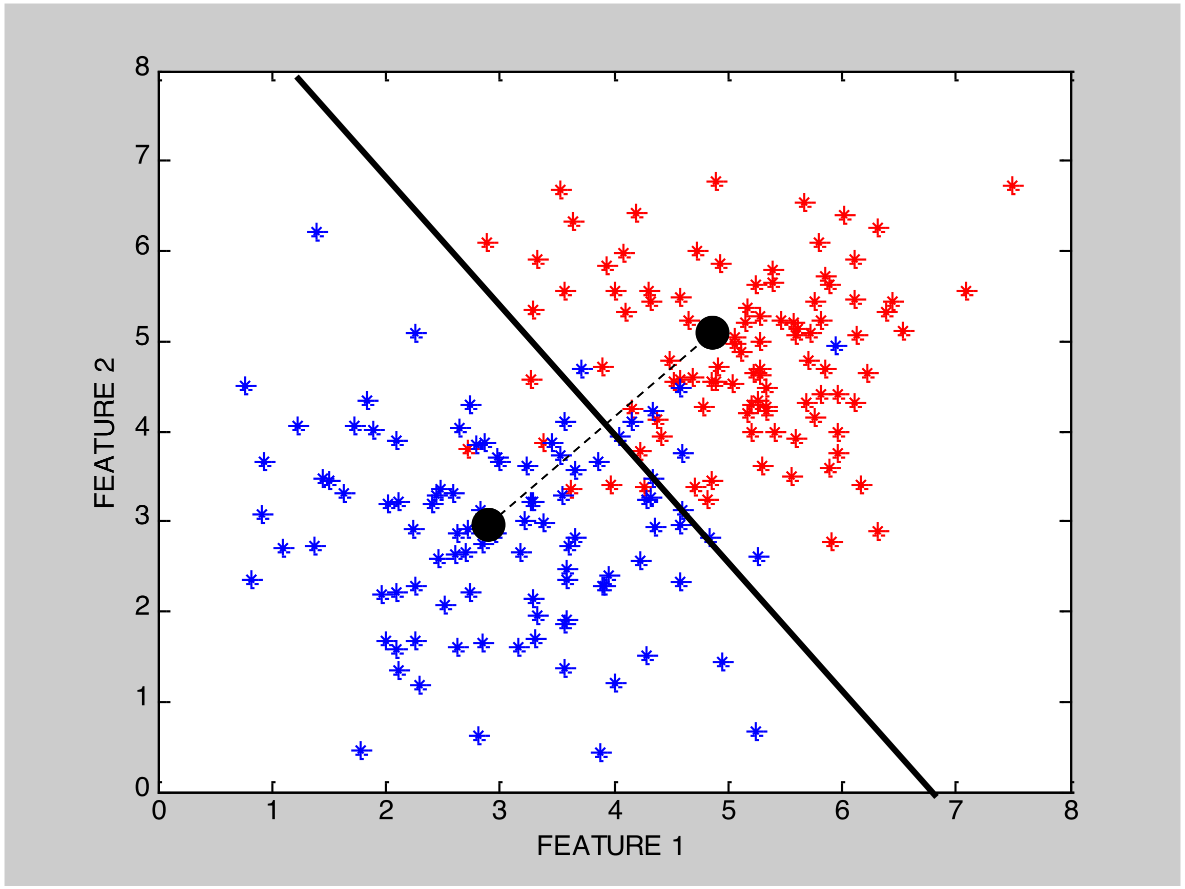


Note: tree boundaries are linear and axis-parallel

A Simple Classifier: Minimum Distance Classifier

- Training
  - Separate training vectors by class
  - Compute the mean for each class, $\underline{\mu}_k$,   k = 1,… m
- Prediction
  - Compute the closest mean to a test vector $\underline{x}'$ (using Euclidean distance)
  - Predict the corresponding class
- In the 2-class case, the decision boundary is defined by the locus of the hyperplane that is halfway between the 2 means and is orthogonal to the line connecting them

Minimum Distance Classifier



Another Example: Nearest Neighbor Classifier

- The nearest-neighbor classifier
  - Given a test point $\underline{x}'$, compute the distance between $\underline{x}'$ and each input data point
  - Find the closest neighbor in the training data
  - Assign $\underline{x}'$ the class label of this neighbor

- The nearest neighbor classifier results in piecewise linear decision boundaries

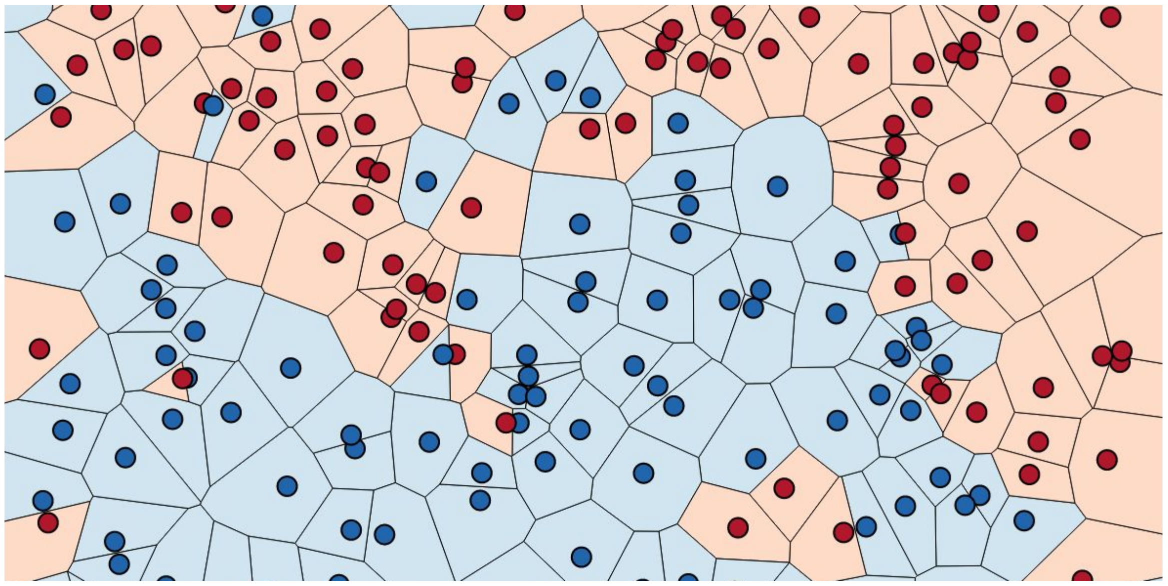
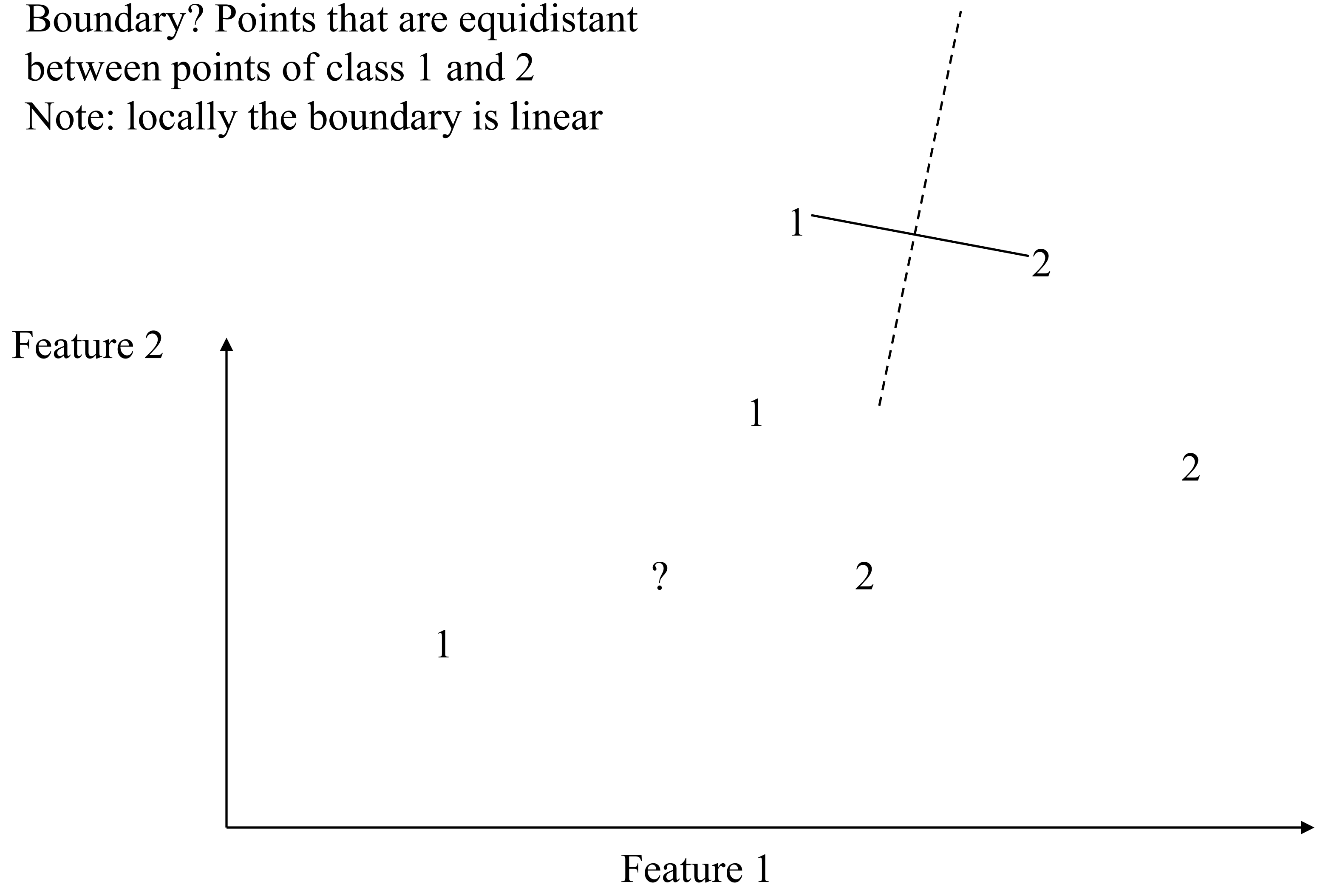


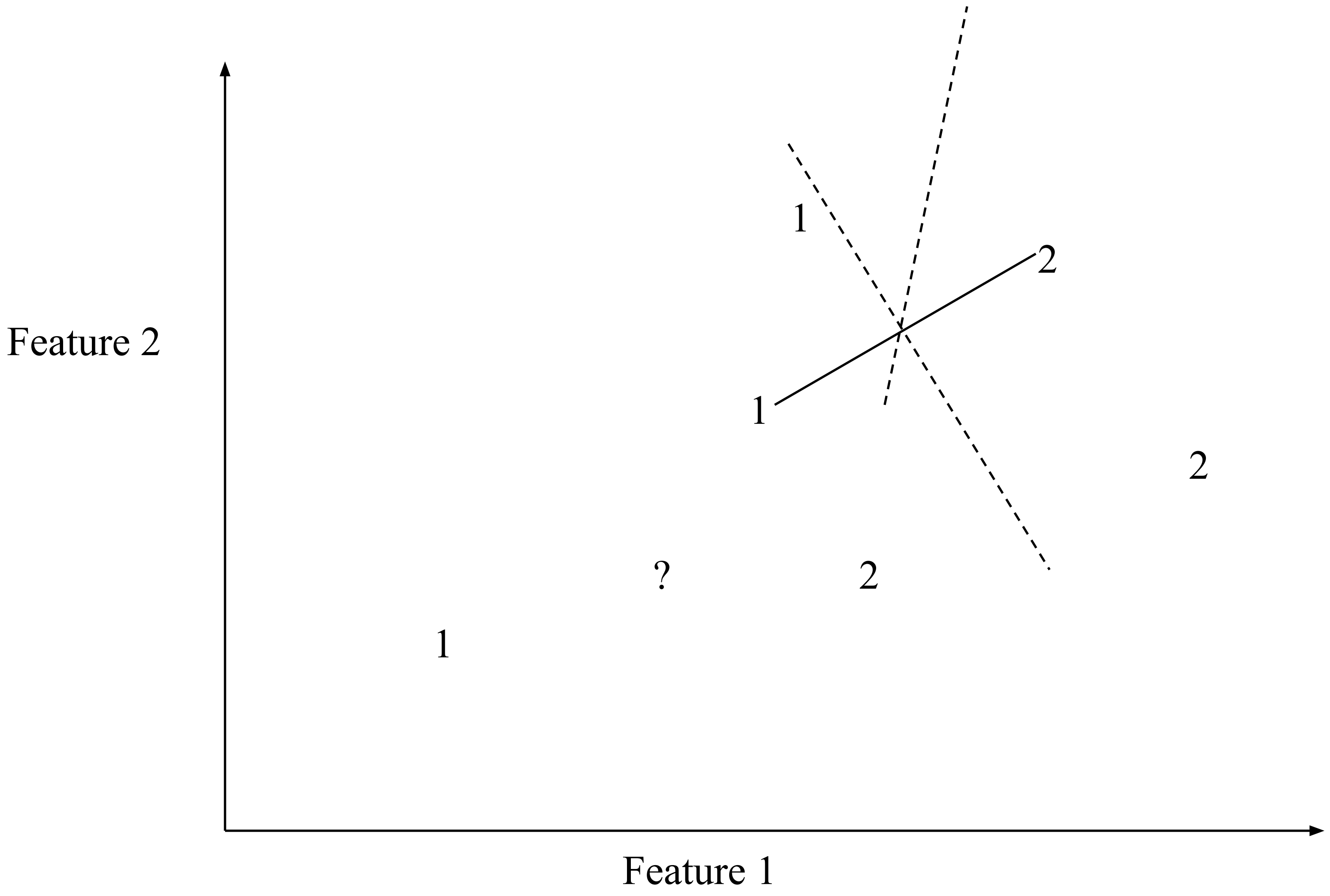
Image Courtesy: http://scott.fortmann-roe.com/docs/BiasVariance.html

Local Decision Boundaries

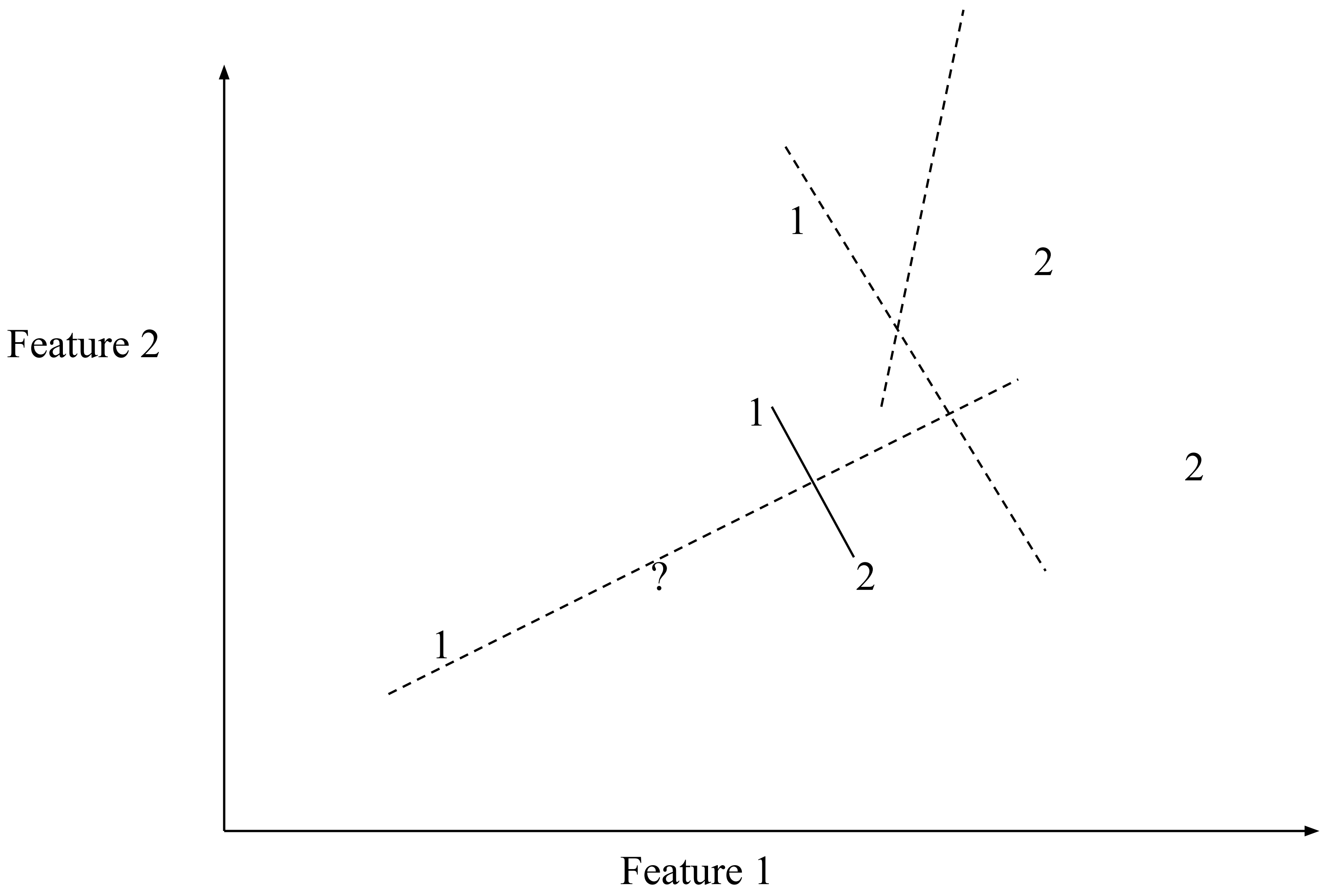
Boundary? Points that are equidistant between points of class 1 and 2
Note: locally the boundary is linear



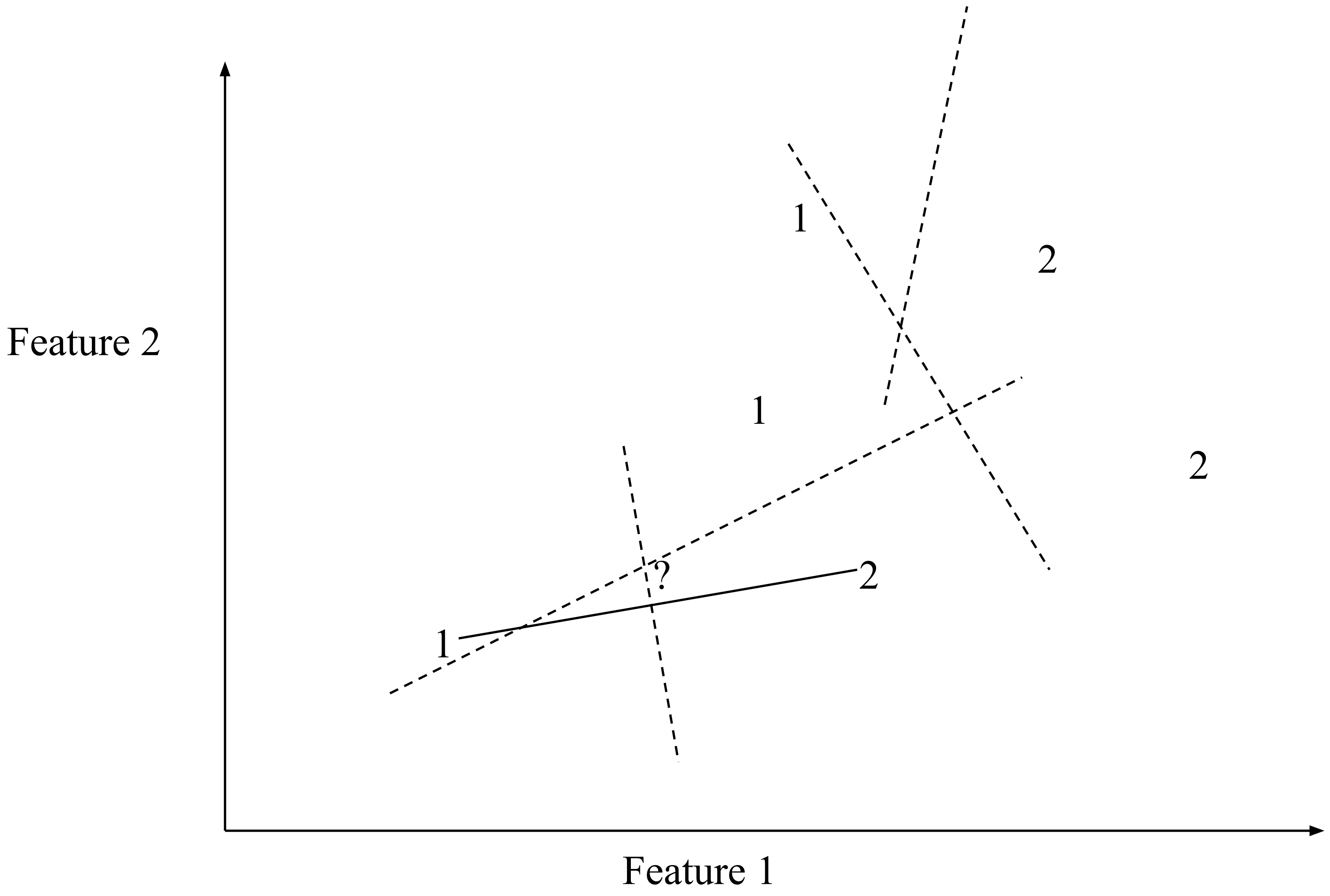
Finding the Decision Boundaries



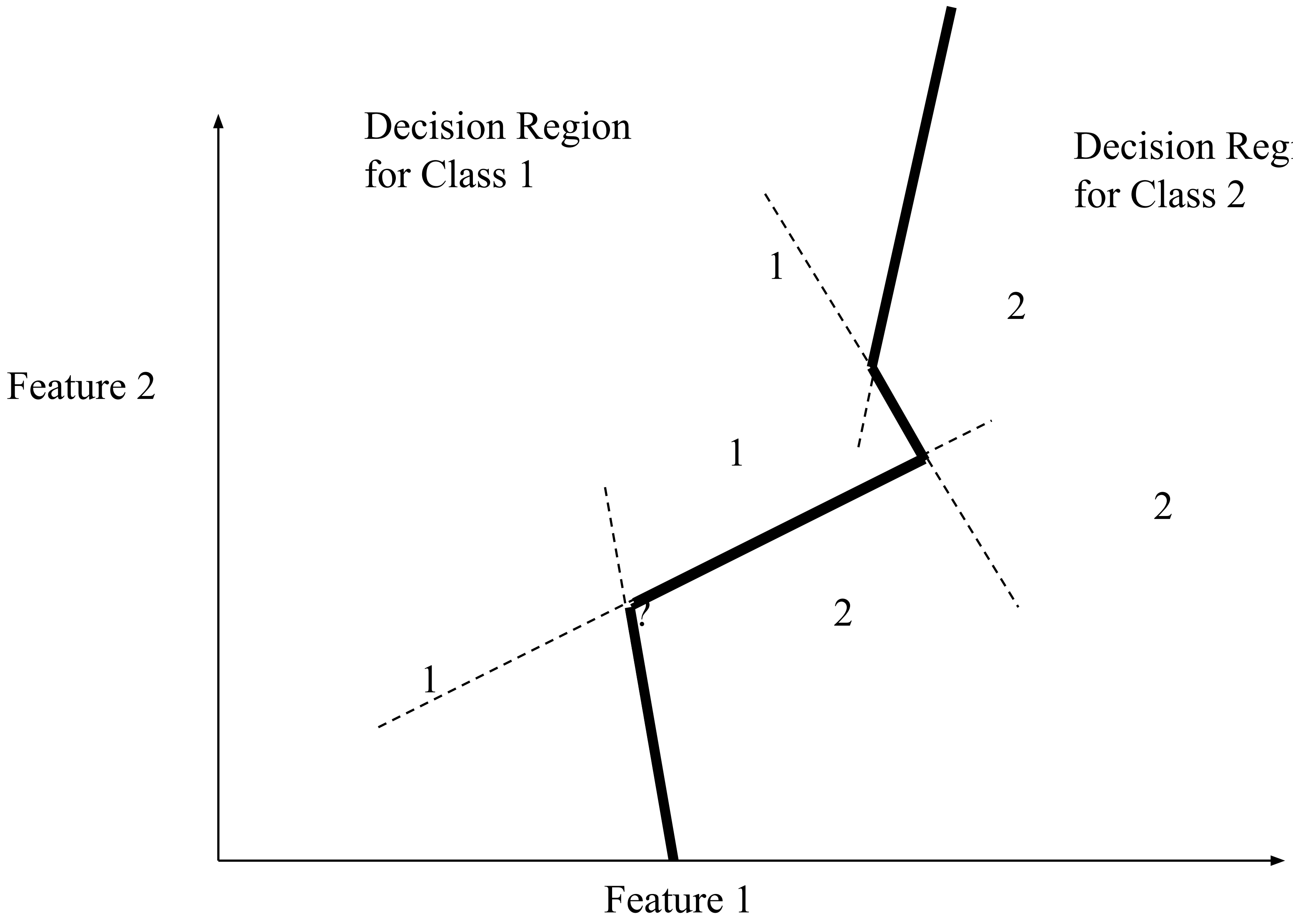
Finding the Decision Boundaries



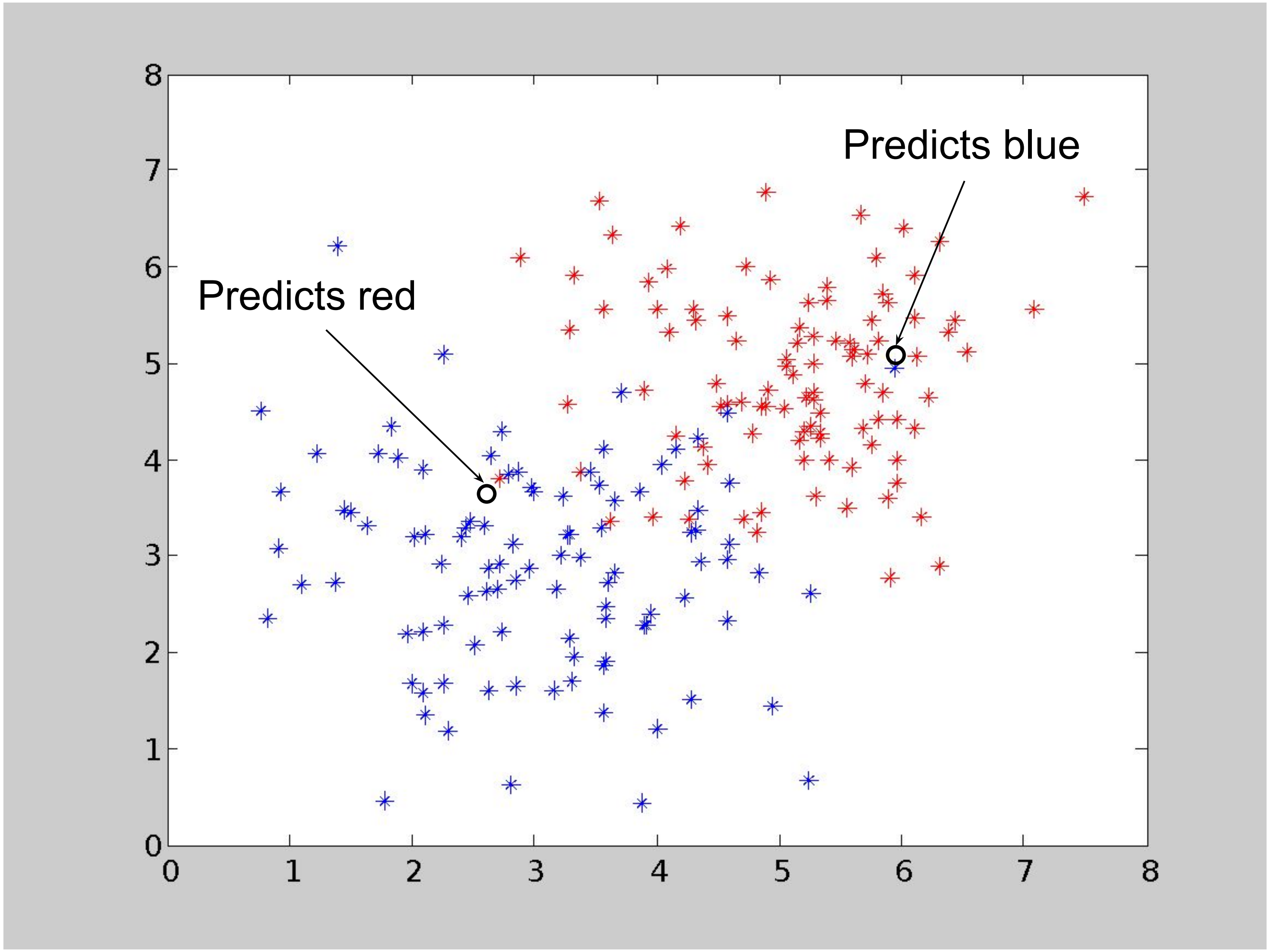
Finding the Decision Boundaries



Overall Boundary = Piecewise Linear

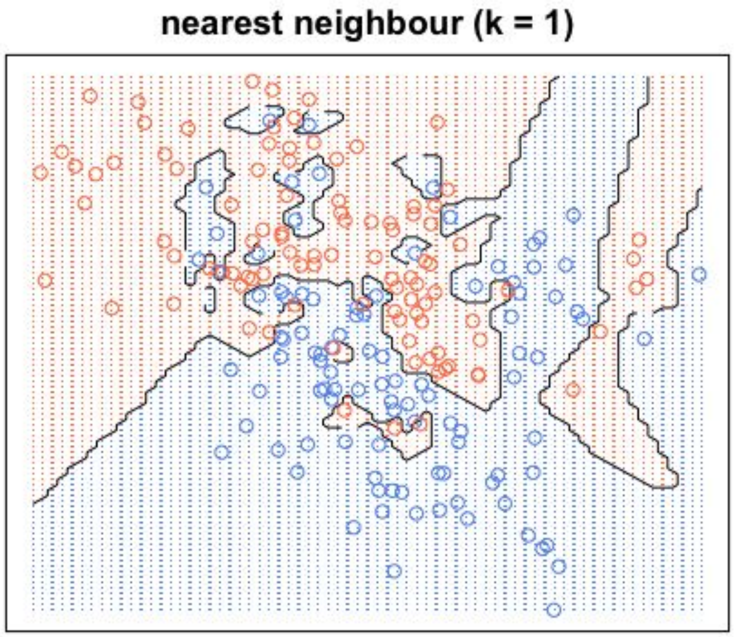


Nearest-Neighbor Boundaries on this data set?



K-Nearest Neighbor Classifier

- Instead of finding the 1 closest neighbors, find k closest neighbors.
- For categorical class labels, take vote based on k-nearest neighbors.
- k can be chosen by cross-validation



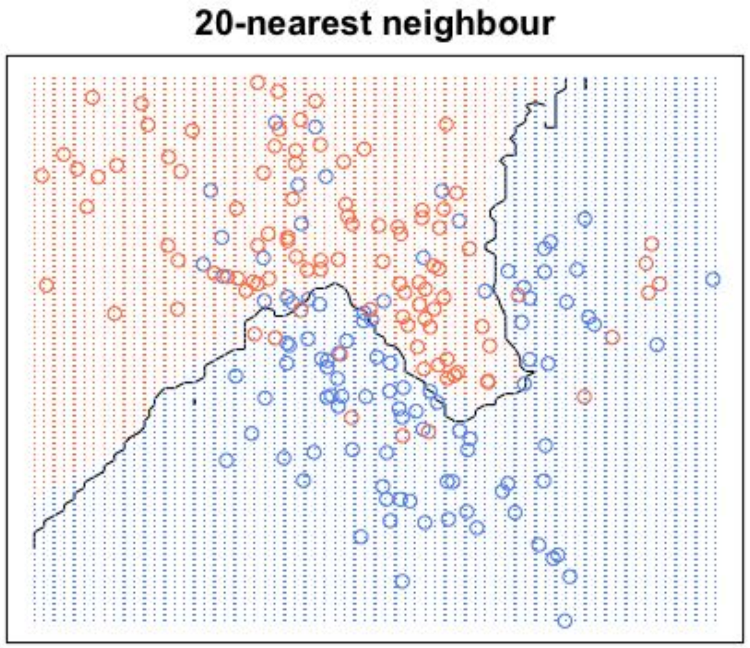
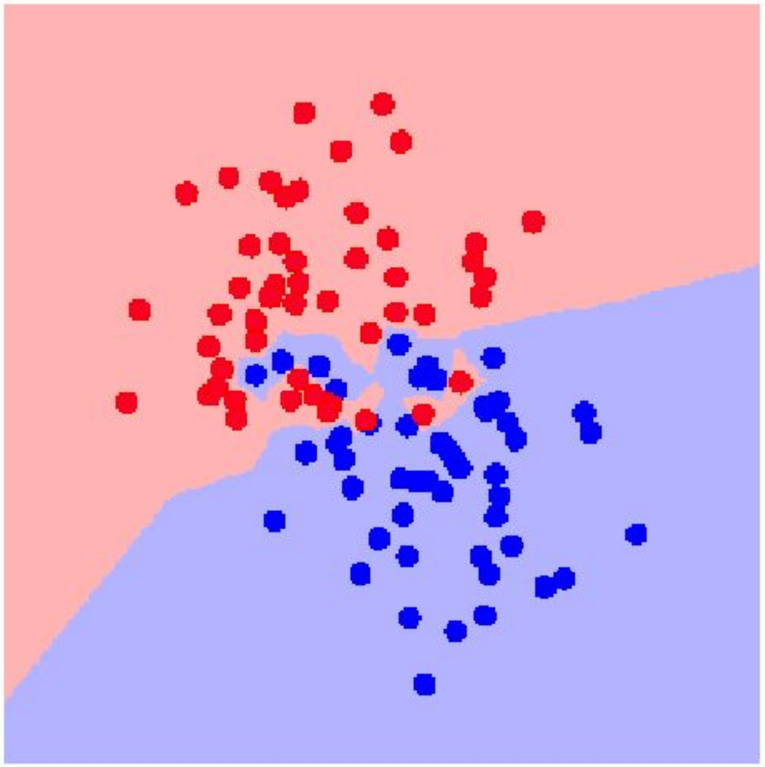


Image Courtesy: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

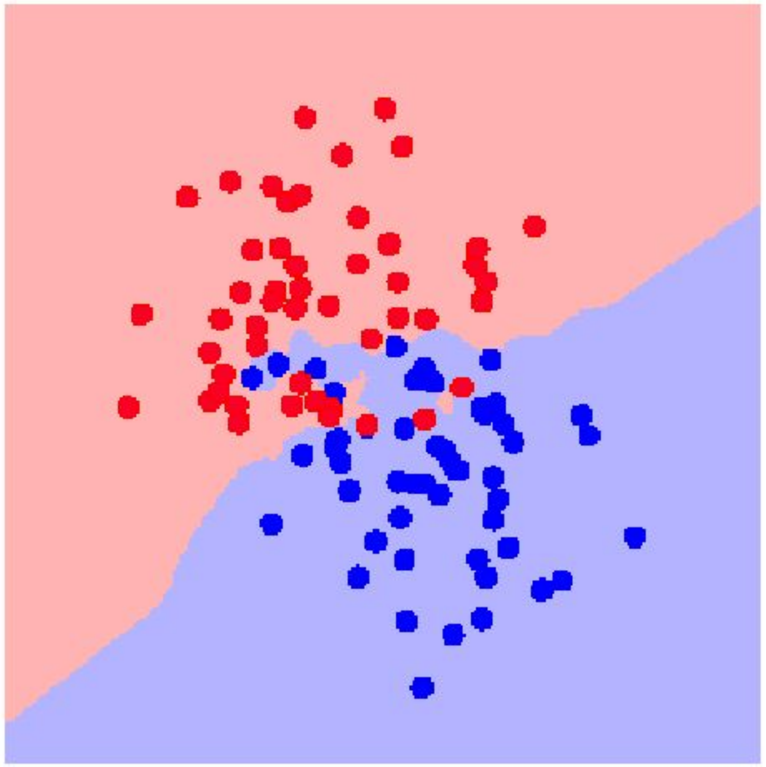
kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

K = 1



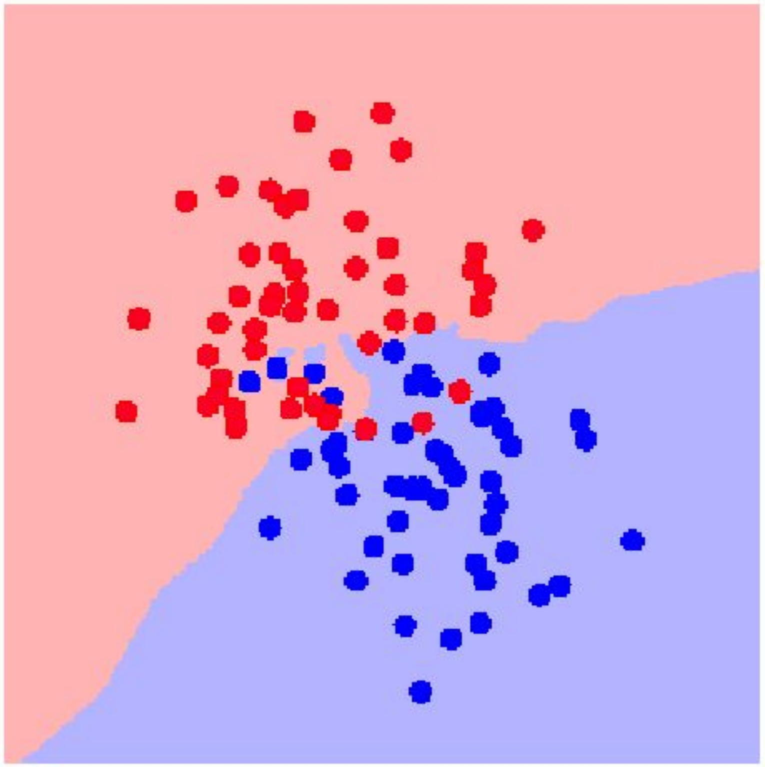
K = 3



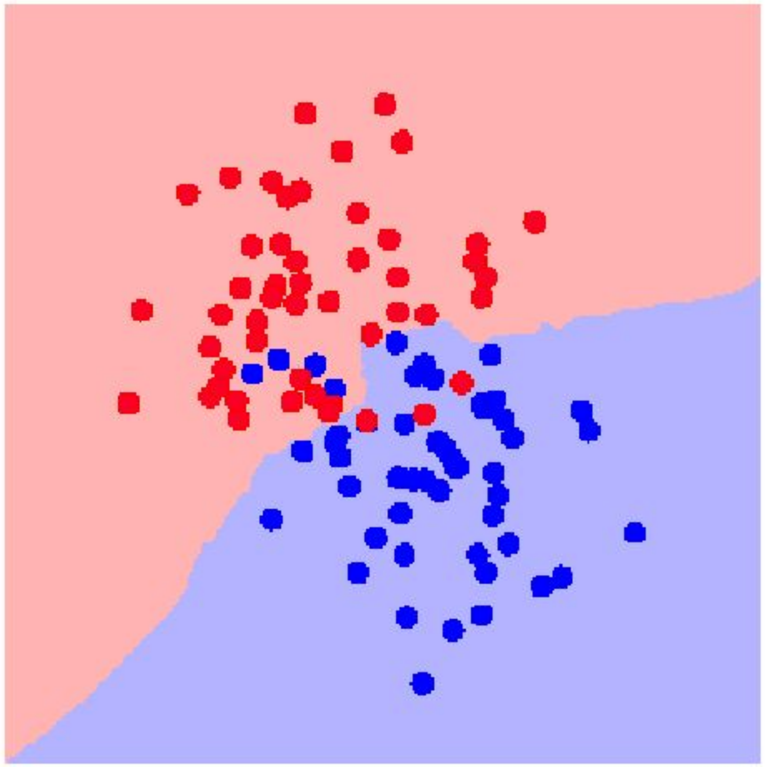
kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

K = 5



K = 7

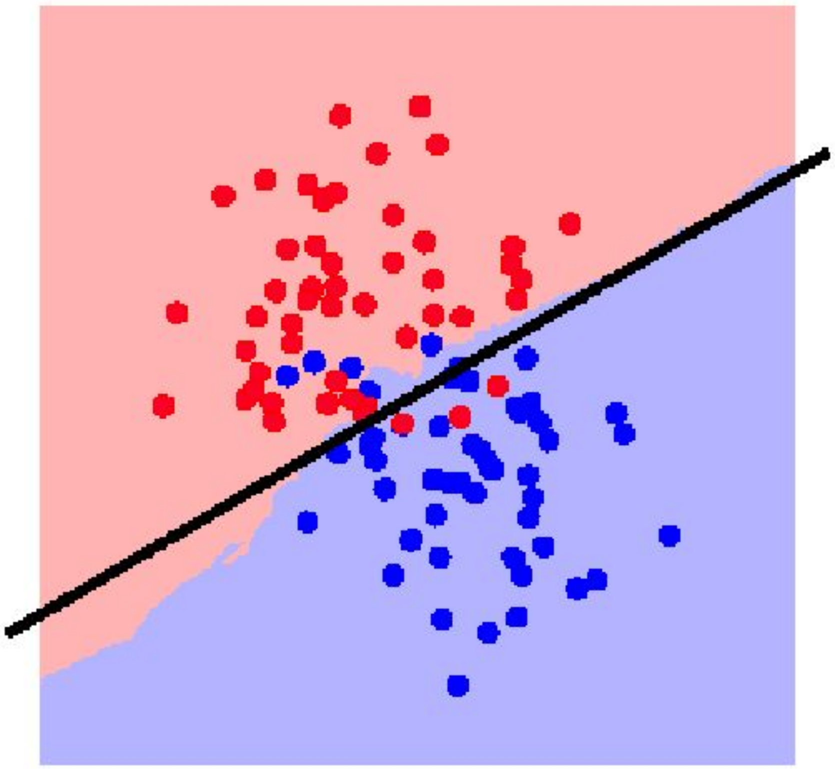


kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

K = 25

Larger K $\Longrightarrow$ Smoother boundary



The kNN Classifier

- The kNN classifier often works very well.
- Easy to implement.
- Easy choice if characteristics of your problem are unknown.

- Can be sensitive to the choice of distance metric.
  - Often normalize feature axis values, e.g., z-score or [0, 1]
    - E.g., if one feature runs larger in magnitude than another

- Can encounter problems with sparse training data.

- Can encounter problems in very high dimensional spaces.
  - Most points are neighbors of most other points.

Linear Classifiers

- Linear classifiers classification decision based on the value of a linear combination of the characteristics.
  - Linear decision boundary (single boundary for 2-class case)

- We can represent a linear decision boundary by a linear equation:

$$w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \sum_j w_j x_j = w^T x = 0$$

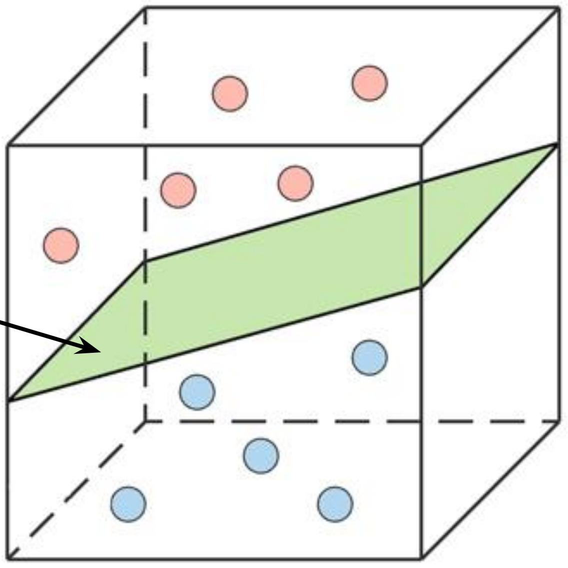
- w_i are the weights (parameters of the model)

Linear Classifiers

$$w_1x_1 + w_2x_2 + \ldots + w_dx_d = \sum_j w_jx_j = w^Tx = 0$$

- This equation defines a ***hyperplane*** in d dimensions
  - A **hyperplane** is a subspace whose dimension is one less than that of its ambient space.
  - If a space is 3-dimensional, its hyperplanes are the 2-dimensional planes; if a space is 2-dimensional, its hyperplanes are the 1-dimensional lines.

A hyperplane in a 3-dimensional space.

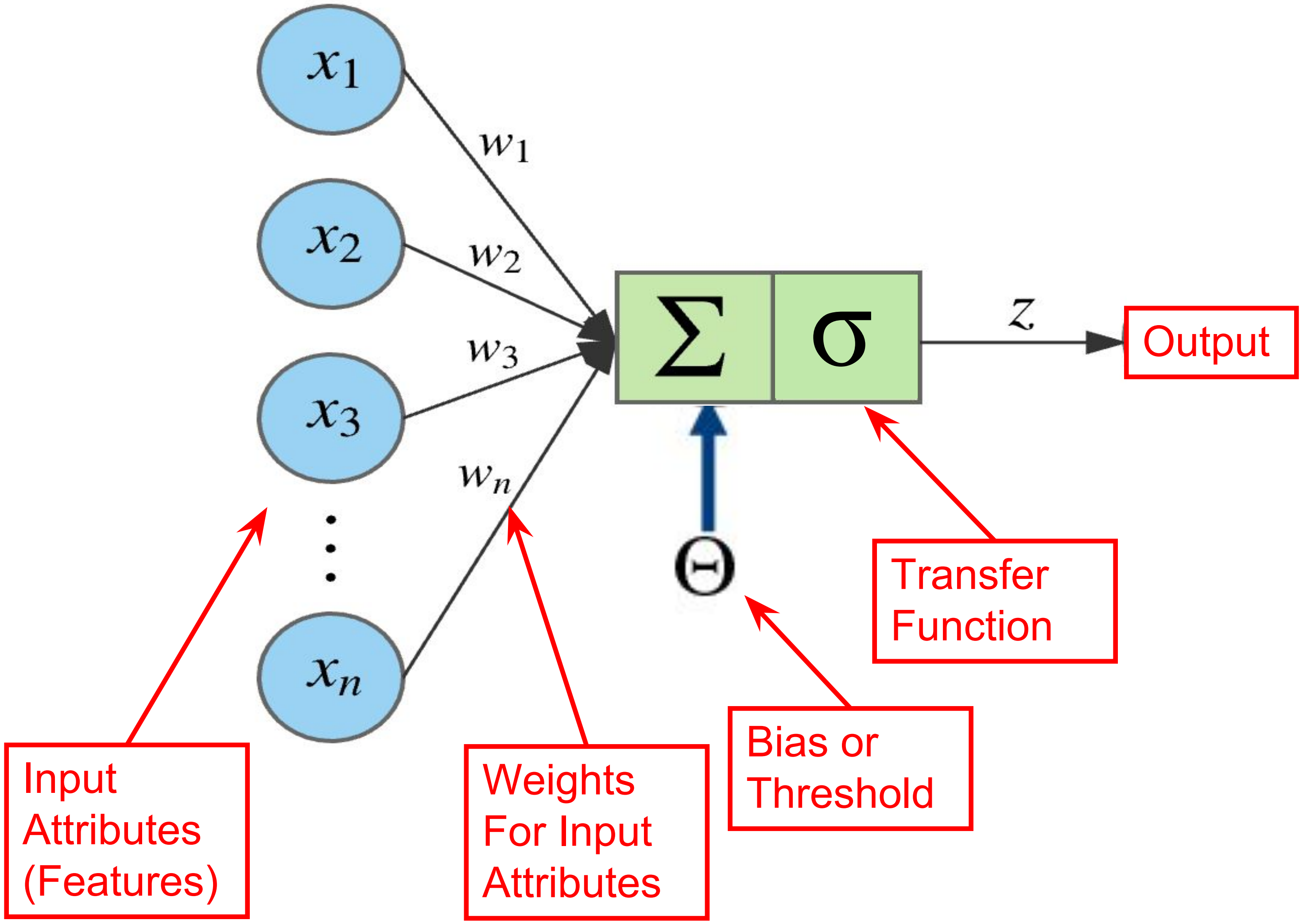


https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Linear Classifiers

- For prediction we simply see if $\sum_j w_j x_j > 0$ for new data x.
- Learning consists of searching in the d-dimensional weight space for the set of weights (the linear boundary) that minimizes an error measure
- A threshold can be introduced by a “dummy” feature that is always one; its weight corresponds to (the negative of) the threshold
- Note that a minimum distance classifier is a special case of a linear classifier

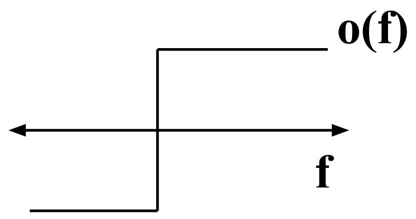
The Perceptron Classifier (pages 729-731 in text)



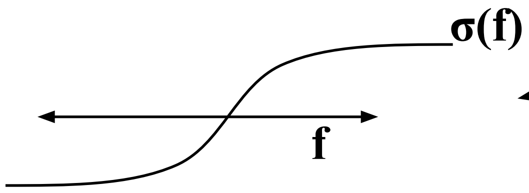
https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Two different types of perceptron output

x-axis below is f(x) = f = weighted sum of inputs
y-axis is the perceptron output



Thresholded output (step function), takes values +1 or -1



Sigmoid output, takes real values between -1 and +1

The sigmoid is in effect an approximation to the threshold function above, but has a gradient that we can use for learning

- Sigmoid function is defined as

$$\sigma[f] = [2 / (1 + \exp[- f])] - 1$$

- Derivative of sigmoid

$$\partial\sigma/\delta f [f] = .5 * (\sigma[f]+1) * (1-\sigma[f])$$

Squared Error for Perceptron with Sigmoidal Output

- Squared error = $E[w] = \sum_i [\sigma(f[\underline{x}(i)]) - y(i)]^2$

  where x(i) is the i-th input vector in the training data, i=1,..N
  y(i) is the ith target value (-1 or 1)

  $f[\underline{x}(i)] = \sum_j w_j x_j$ is the weighted sum of i-th inputs

  $\sigma(f[\underline{x}(i)])$ is the sigmoid of the weighted sum

- Note that everything is fixed (once we have the training data) except for the weights w

- So we want to minimize E[w] as a function of w

Gradient Descent Learning of Weights

Gradient Descent Rule:

$$\underline{\mathbf{w}}_{\text{new}} = \underline{\mathbf{w}}_{\text{old}} - \alpha \, \Delta (E[\underline{\mathbf{w}}])$$

where

$\Delta (E[\underline{w}])$ is the gradient of the error function E wrt weights, and

α is the learning rate (small, positive)

Notes:

1. This moves us downhill in direction $\Delta (E[\underline{w}])$ (steepest downhill)

2. How far we go is determined by the value of α

Pseudo-code for Perceptron Training

```
Initialize each $w_j$ (e.g.,randomly)

While (termination condition not satisfied)
    for i = 1: N     % loop over data points (an iteration)
        for j= 1 : d     % loop over weights
            $\underline{w}_{j,\ new} = \underline{w}_j - \alpha\ \Delta(E[\underline{w}_j])$
        end
    calculate termination condition
    end
```

- Inputs: N features, N targets (class labels), learning rate η
- Outputs: a set of learned weights

Comments on Perceptron Learning

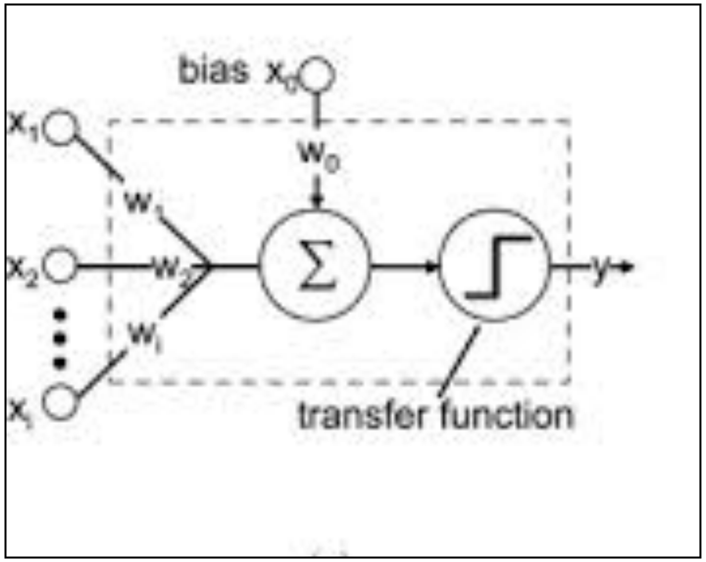
- Iteration = one pass through all of the data

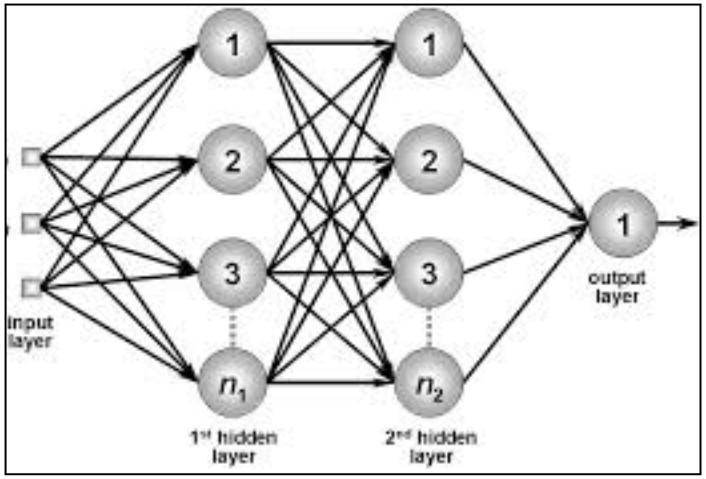
- Algorithm presented = incremental gradient descent
  - Weights are updated after visiting each input example
  - Alternatives
    - Batch: update weights after each iteration (typically slower)
    - Stochastic: randomly select examples and then do weight updates

- Rate of convergence
  - E[$\underline{w}$] is convex as a function of $\underline{w}$, so no local minima
  - Convergence is guaranteed as long as learning rate is small enough
    - But if we make it too small, learning will be *very* slow
  - If learning rate is too large, we move further, but can overshoot the solution and oscillate, and not converge at all

Multi-Layer Perceptrons (Artificial Neural Networks)

(sections 18.7.3-18.7.4 in textbook)





Multi-Layer Perceptrons (Artificial Neural Networks)
(sections 18.7.3-18.7.4 in textbook)

- What if we took K perceptrons and trained them in parallel and then took a weighted sum of their sigmoidal outputs?
  - This is a multi-layer neural network with a single "hidden" layer (the outputs of the first set of perceptrons)

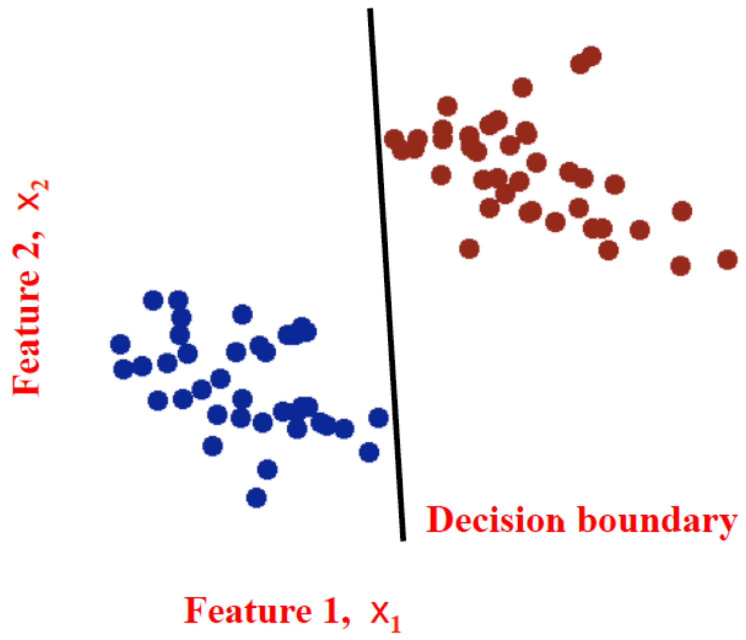
- How would we train such a model?
  - Backpropagation algorithm = clever way to do gradient descent
  - Bad news: many local minima and many parameters
    - training is hard and slow
  - Good news: can learn general non-linear decision boundaries

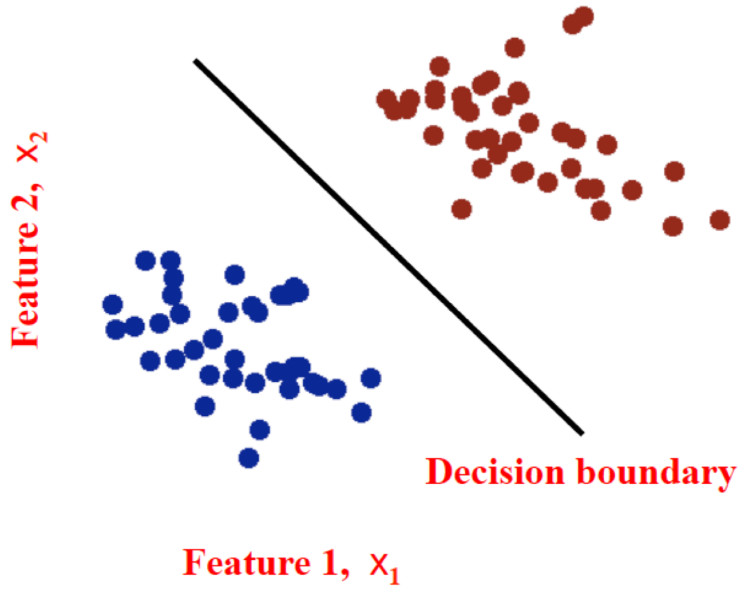
- Generated much excitement in AI in the late 1980's and 1990's

- New current excitement with very large "deep learning" networks

Which decision boundary is "better"?

- Both have zero training error (perfect training accuracy).
- But one seems intuitively better...





Support Vector Machines (SVM): "Modern perceptrons" (section 18.9, R&N)

- A modern linear separator classifier
  - Essentially, a perceptron with a few extra wrinkles

- Constructs a **"maximum margin separator"**
  - A linear decision boundary with the largest possible distance from the decision boundary to the example points it separates
  - "Margin" = Distance from decision boundary to closest example
  - The "maximum margin" helps SVMs to generalize well

- Can embed the data in a non-linear higher dimension space
  - Transform data into higher dimensional space
  - Constructs a linear separating hyperplane in that space
    - **This can be a non-linear boundary in the original space**

- **Currently most popular "off-the shelf" supervised classifier.**

Can embed the data in a non-linear higher dimension space

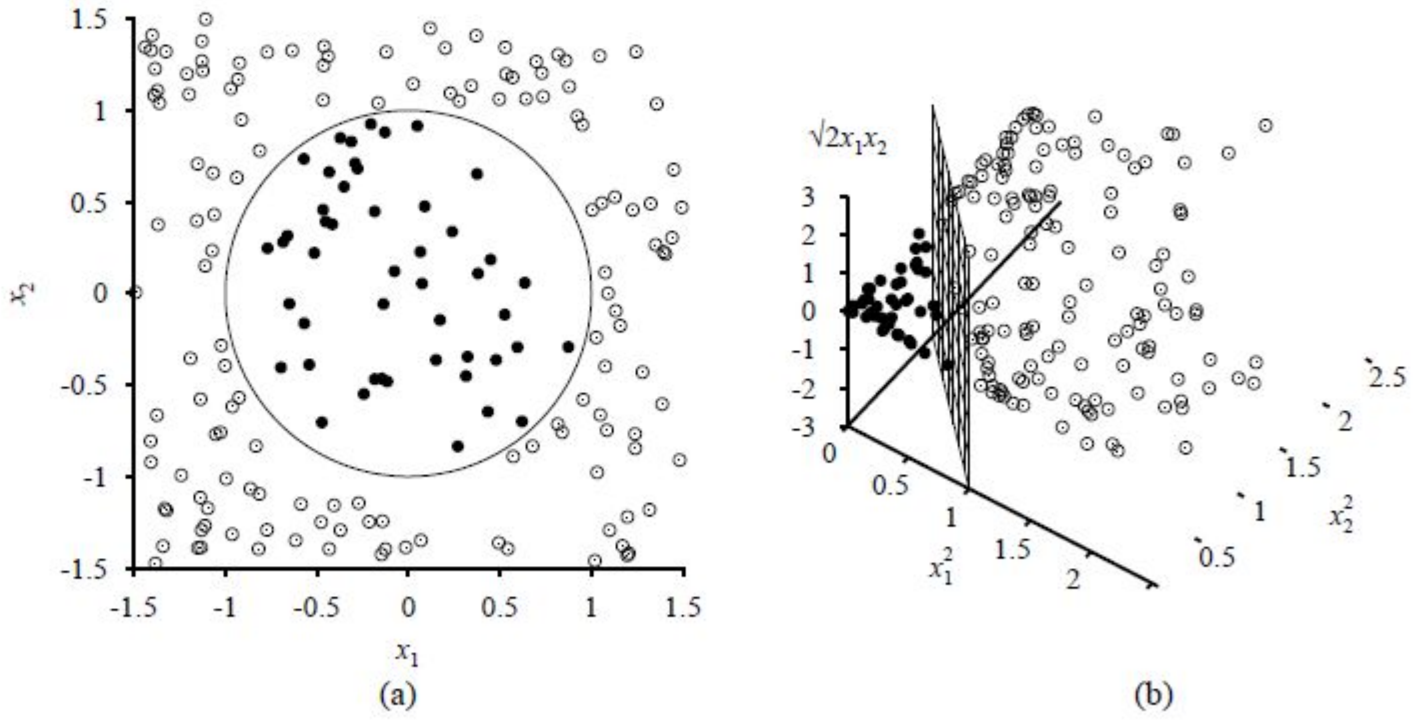
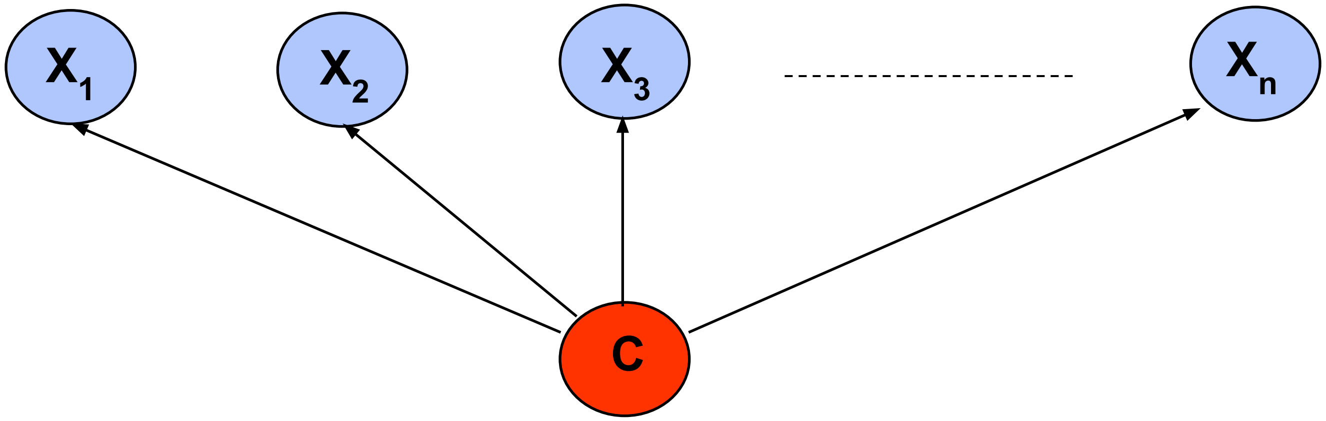


Figure 18.31 **FILES: .** (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions. Figure 18.29(b) gives a closeup of the separator in (b).

Naïve Bayes Model (section 20.2.2 R&N 3rd ed.)



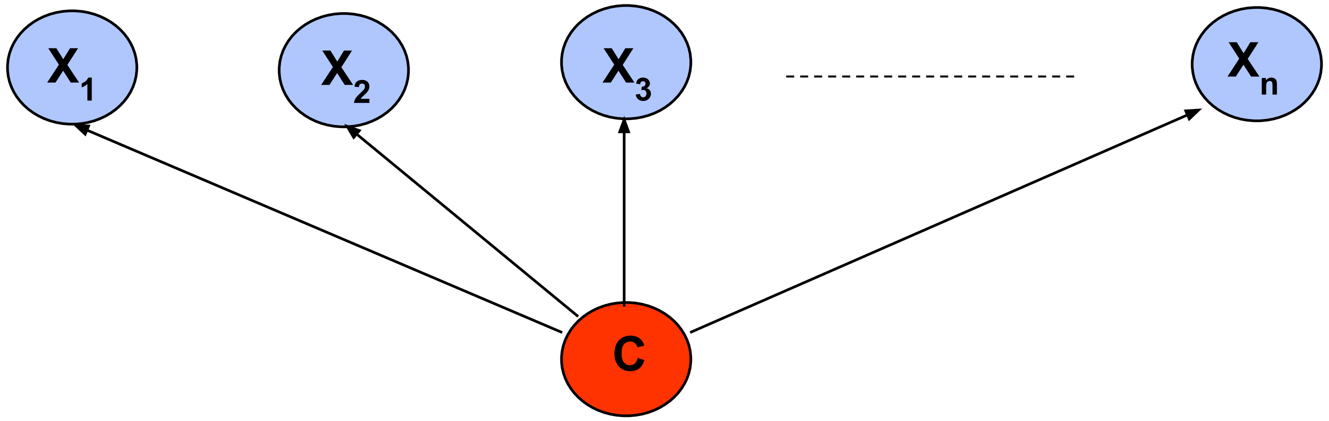
Goal: We want to estimate $P(C \mid X_1,\ldots X_n)$

Solution: Use Bayes' Rule to turn $P(C \mid X_1,\ldots X_n)$ into a proportionally equivalent expression that involves only $P(C)$ and $P(X_1,\ldots X_n \mid C)$.

Then assume that feature values are ***conditionally independent*** given class, which allows us to turn $P(X_1,\ldots X_n \mid C)$ into $\Pi_i\ P(X_i \mid C)$.

We estimate $P(C)$ easily from the ***frequency*** with which each class appears within our training data, and we estimate $P(X_i \mid C)$ easily from the frequency with which each X_i appears in each class C within our training data.

Naïve Bayes Model (section 20.2.2 R&N 3rd ed.)



Bayes Rule: $P(C \mid X_1,\ldots X_n)$ is proportional to $P(C) \prod_i P(X_i \mid C)$
[note: denominator $P(X_1,\ldots X_n)$ is constant for all classes, may be ignored.]

Features Xi are conditionally independent given the class variable C

- choose the class value c_i with the highest $P(c_i \mid x_1,\ldots, x_n)$
- simple to implement, often works very well
- e.g., spam email classification: X's = counts of words in emails

Conditional probabilities $P(X_i \mid C)$ can easily be estimated from labeled date

- Problem: Need to avoid zeroes, e.g., from limited training data
- Solutions: Pseudo-counts, beta[a,b] distribution, etc.

Summary

- Supervised Machine Learning
  - Given a labeled training data set, a class of models, and an error function, this is essentially a search or optimization problem

- Different Machine Learning classifiers & their decision boundaries.
  - Decision trees
  - K-nearest neighbors
  - Perceptrons
  - Support vector Machines (SVMs),
  - Neural Networks
  - Naïve Bayes