Classifiers

CS 510

Lecture #18

April 18th, 2013

Colorado State University

Programming Assignment #3

- Due today
  - Any questions?
  - How is it going?

Where are we?

- Two general approaches to recognition
  – Constellations
  – Bag of Features
- Each require a classifier
  – So let's talk about classifiers

Note: we will surf through material that makes up most of CS540 and CS545, and part of CS548

Basic Idea : Generalization

Training Samples:

Label	F1	F2	F3	F4	F5
True	3.6	3.0	-1.2	9.8	-0.5
True	2.0	1.0	0.9	1.3	3.1
False	-1.3	-4.1	0.8	1.1	-8.0
True	-1.4	-3.9	0.1	1.2	2.5
False	-9.0	-2.6	-0.5	1.0	1.1

Test Samples:

Label	F1	F2	F3	F4	F5
???	-3.6	0.3	0.1	-4.4	0.9

Visualization

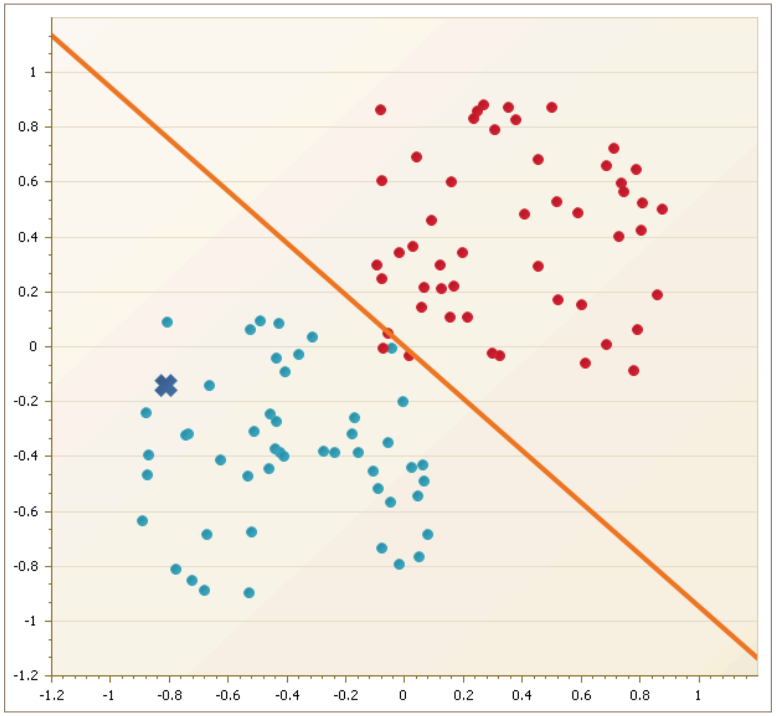
- Samples as points in a feature space
  - Example from last slide : 5 dimensional
- Label as color (or symbol)
  - Example : red for true, black for false
- Goal: segment feature space
  - Every region contains samples of one label
  - All of feature space in some region
  - New samples assigned label of their region

Classifiers : methods of segmenting feature space

- Perceptrons (linear)
- Support Vector Machines (linear)
- Multi-layer feed-forward networks (differentiable functions)
- Bayesian (elliptical)
- Decision Trees (recursive linear)
- Nearest Neighbor (multiple ellipses)

Perceptrons

- The first machine learning algorithm
- Perceptrons divide linearly separable data



Goal: find the parameters of a hyper-plane that separates the classes

Colorado State University

Perceptrons (II)

- Definition :

$$f(x) = \begin{cases} 1 \text{ if } \mathrm{w} \cdot \mathrm{x} + \mathrm{b} > 0 \\ 0 \text{ otherwise} \end{cases}$$

Where x is a data sample (feature vector)

w is the learned weight vector

b is a learned bias term (why?)

Perceptron Learning Rule

- $D = \{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)\}$, where d_i are the desired labels (0/1).
- Algorithm:
  - Randomly initialize w & b (small values)
  - Iteratively update the weights until convergence

$$w_i(t+1) = w_i(t) + \alpha\left(d_j - f(x_j)\right)x_{i,j}$$

  - Same for bias term b

Why does this work?

- Correctly labeled instances do not change the weights
  - $(d_i - f(x_i)) = 0$
- Incorrect instances
  - Increment weights if $d_i > f(x_i)$
  - Decrement weights if $f(x_i) < d_i$
  - By an amount that depends on $x_{i,j}$

Problems with Perceptrons

- Limited to linear divisions
  - Converges if data is linearly separable
  - Otherwise, use a difference threshold to terminate algorithm
- Expensive : O(NDC)
  - N = number of samples
  - D = number of feature dimensions
  - C = number of cycles (iterations)

Better Linear Classifier: SVM

- Like perceptron, produces a weight vector and bias term
- Unlike perceptron
  - Maximizes the margin between classes
    - Distance between hyper-plane and nearby samples
  - Updates ignore samples far from boundary
    - More efficient
      - Helps when D is large
    - Samples far from boundary do not influence its angle
  - For the math of how it does this, take CS548

Key to SVMs : Kernels

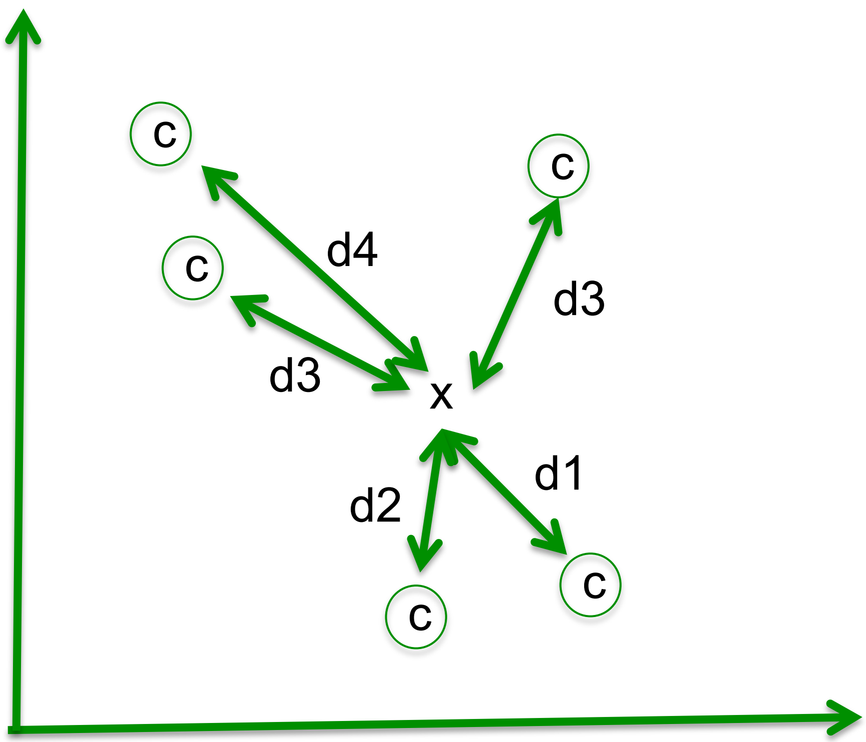
- Linear classifiers are limited
  - Data typically isn't linearly separable
- Unless you project into a higher dimensional space
  - Linear functions in higher dimensions may be complex functions in the original feature space
- Simple example
  - Given D features, create D^2+D dimensional vectors
  - Contains all D original features, plus
  - $X_i X_j$ for all i, j
  - 2^{nd}-order functions in the original space are linear in the expanded space!

Radial Basis Kernels

- Radial basis functions (RBFs) are functions whose value depends on distance from a point
  – Typically Gaussian, e.g. $f\left(x\right) \approx e^{\frac{-|x-c|}{\sigma}}$

- Given N training samples, create N RBFs, one centered at each sample
- Convert samples into points in N dimensions, where each dimension is a distance to a training sample

Kernelized Data



Data in original 2D data space

Data in 5D Kernel Space

Multi-layer Perceptrons

- Another approach is to directly learn non-linear boundaries in the original feature space
- Perceptrons are linear, but you can add a sigmoid function:

$$\varphi(v) = \left(1 + e^{-v}\right)^{-1}$$

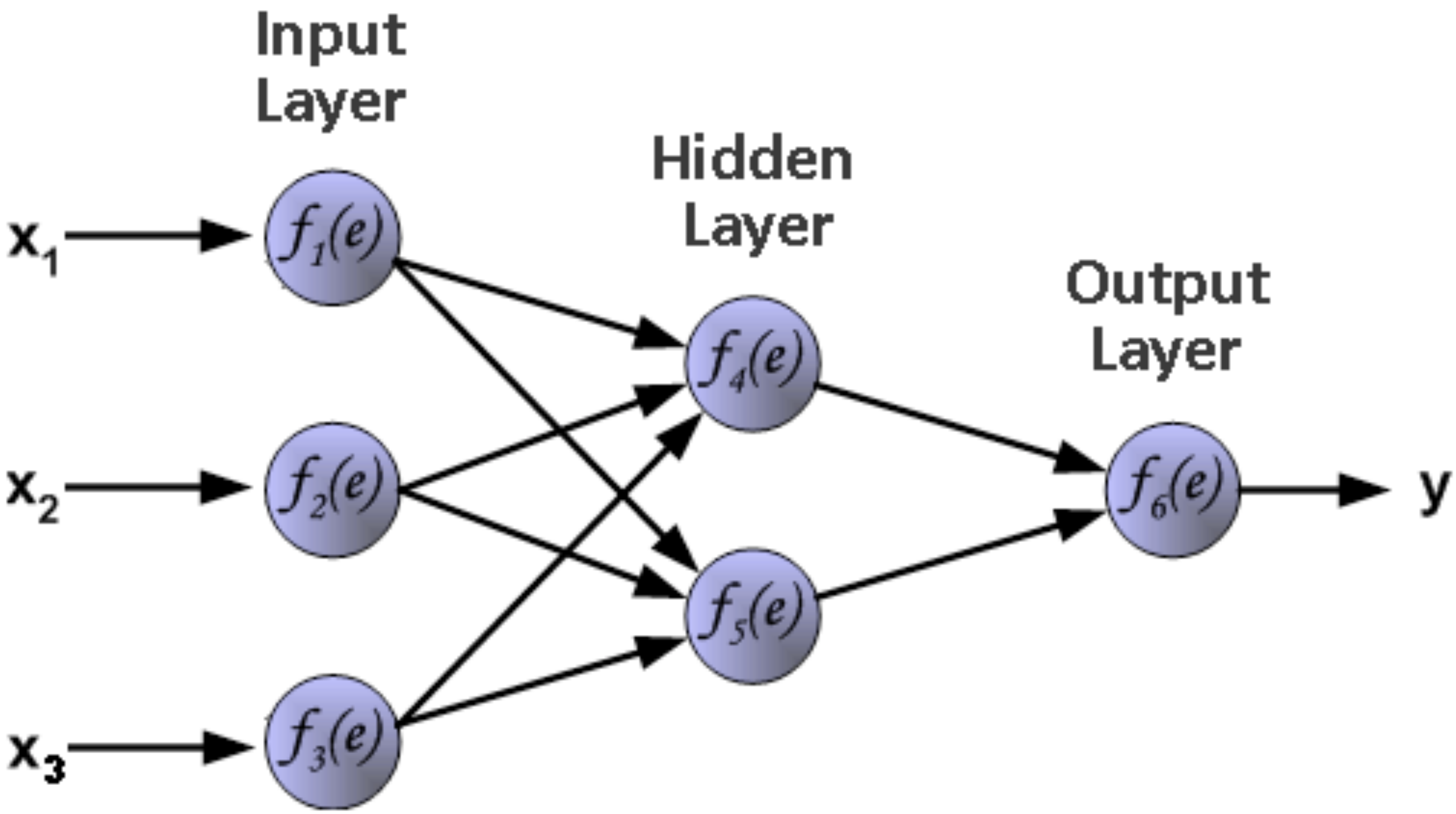
$$g(x) = \varphi\left(f(x)\right) = \varphi(w \cdot x + b)$$

- Perceptrons + sigmoid are non-linear

Multi-layer Perceptrons (II)

- Create multiple perceptrons (with sigmoids)
- Feed their outputs into another perceptron
  - Non-linear combination of non-linear functions
- Backpropagation training rule
  - Minimizes the squared sum of errors
  - Computes the derivative of each weight/bias for each training sample
  - Iteratively alters weights to minimize the errors

Illustration of a Multi-layer Perceptron



Backpropation

- Every sample comes with its target output
- Minimize the squared error:

$$E = \sum_i \left(d_i - f\left(x\right)\right)^2$$

- By computing the partial derivative of E with regard to each weight $w_{i,j}$
- Adjust the weights to minimize the error

Backpropagation (II)

$$E = \sum_i \left(d_i - f\left(x_i\right)\right)^2$$

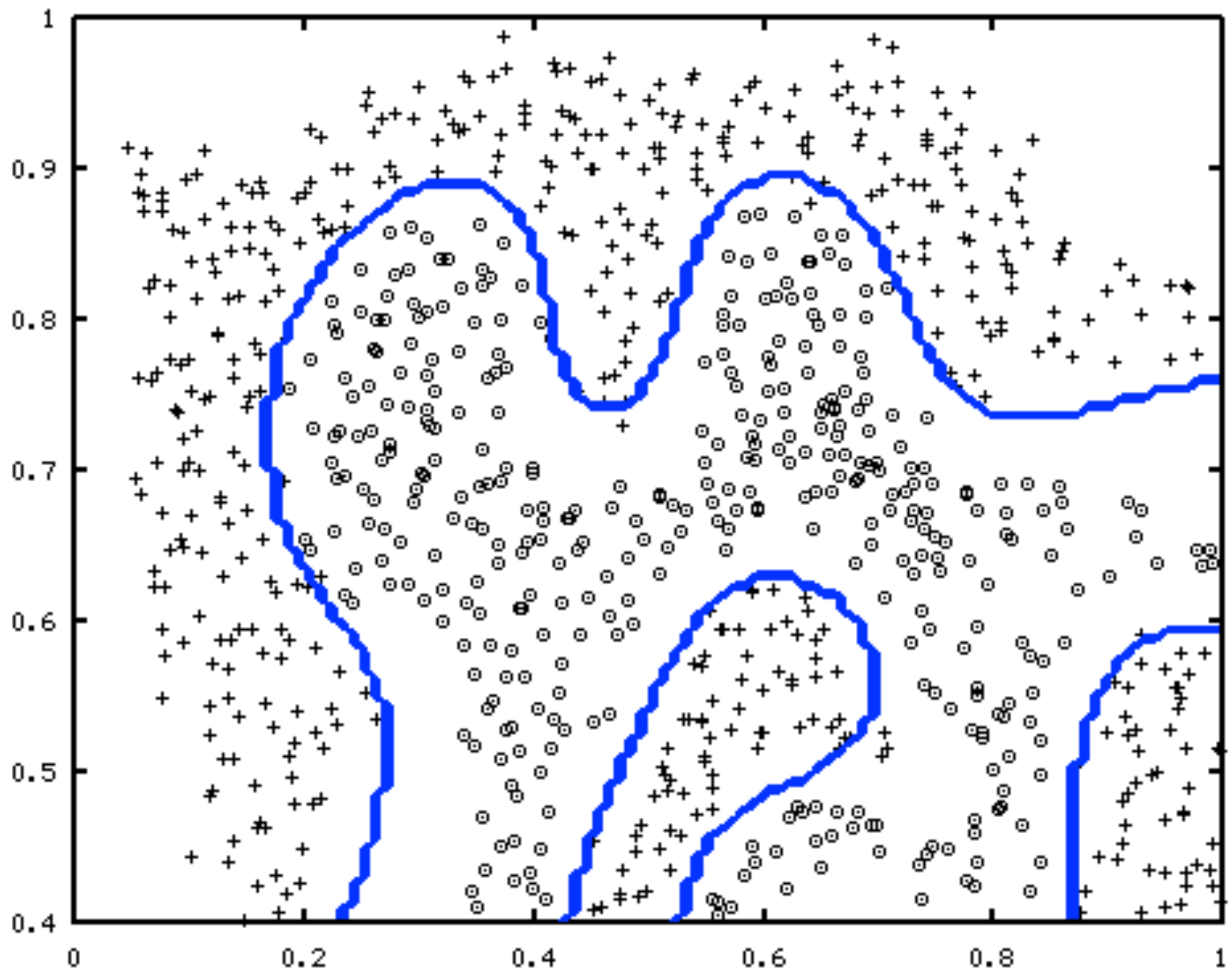
$$E = \sum_i \left(d_i - \varphi\left(w_{h1} y_i + b_h\right)\right)^2$$

$$E = \sum_i \left(d_i - \varphi\left(w_{h1}\left[\varphi\left(w_{i1} x_i + b_{i1}\right), \varphi\left(w_{i2} x_i + b_{i2}\right), \ldots\right] + b_h\right)\right)^2$$

$$\frac{\partial E}{\partial w_{ij}} = \sum_i 2\left(d_i - f\left(x_i\right)\right)\frac{\partial f\left(x_i\right)}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = 2\sum_i \left(d_i - f\left(x_i\right)\right)\left(f\left(x_i\right)\left(1 - f\left(x_i\right)\right)\right) w_{ij}$$

Multi-layer Perceptron Illustration



http://stackoverflow.com/questions/14440658/multi-layer-perceptron-finding-the-separating-curve

Colorado State University