Classification

EE379K - Architectures for Big Data Sciences

Dr. Sriram Vishwanath
Department of Electrical and Computer Engineering
The University of Texas at Austin

Spring 2016

Classification Definition

- Supervised Learning Model
- Given a set of data, we want to classify the data into 2 or more categories
  - emails: classify spam or not spam
  - tumors: classify malignant or benign
  - digital images: classify as person, cat, dog, or monkey
  - Dr. Dimakis' example: fruit: tasty or not tasty
- Classifiers are trained on known data with known labels so that they can predict labels for new data
- The known data is referred to as the **training set**

Types of Classifiers

- Perceptron
- Logistical Regression
- Naive Bayes
- Support Vector Machines
- Random Forests

Simplest Classifier: Perceptron

For an input data point x with **features** x1, x2, and x3, we wish to classify the data point as belonging to one of two classes

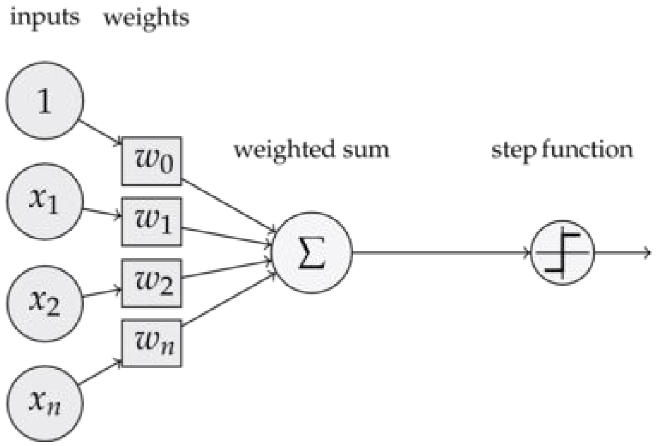
y is the class we assign to x: y ∈ {0,1}

w0 is the **threshold** of decision

w1, w2, and w3 are the **weights** we assign to each feature (how important are they to the decision)

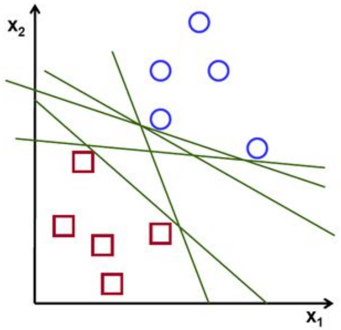
y = 0 if **w·x** <= -w0

y = 1 if **w·x** > -w0



Perceptron

- **w·x** + w0 defines a hyperplane that separates the data points into two categories
- There are many hyperplanes that can divide the data into two categories defined by a perceptron



- **h(w) = w·x** + w0 is called the **hypothesis** function
- It does not have to be linear! (Only for perceptrons)
- Polynomial hypothesis functions
  - h(w) = w0 + w1*x + w2*x^2 + w3*x^3

Classification Measures - Loss functions

- Goal of classification is to find the **optimal** hyperplane that separates the data
  - hyperplane that minimizes some measurement of “**error” or loss**
- Different classifiers minimize different loss functions:
  - 0-1 Loss
  - Squared Loss
  - Logistic Loss
  - Hinge Loss

Classification Goal

- Polynomial hypothesis based classifiers:
- Find weights **w** = <w1, w2, w3, …> so as to minimize the sum of the loss functions for all training samples (x_i ,y_i)

$$\min(\mathbf{w}) \left[\sum L(\mathbf{y}, h(\mathbf{w},\mathbf{x})) \right]$$

- Different classifiers have different forms of **h** (the hypothesis function) and utilize different loss functions **L**

Choosing a Loss Function

- Finding the optimal hyper-curve (hypothesis) for classification for these cases boils down to minimizing a classification loss function
- Basic (unconstrained) Optimization:
  - Analytical (set derivative = 0 and solve for parameters)
    - inefficient for large number of parameters
  - Gradient Descent
    - most widely used
- We want our loss function to be easily differentiable (continuous and smooth) and convex

0-1 Loss Function

- For a hypothesis function h(**w**,**x**), training sample (**x**,**y**)

$$L_{0/1}(\mathbf{h}(\mathbf{w},\mathbf{x}),\mathbf{y}) = 1 \text{ if } |\mathbf{h}(\mathbf{w},\mathbf{x}) - \mathbf{y}| >= \varepsilon$$

$$L_{0/1}(\mathbf{h}(\mathbf{w},\mathbf{x}),\mathbf{y}) = 0 \text{ if } |\mathbf{h}(\mathbf{w},\mathbf{x}) - \mathbf{y}| < \varepsilon$$

- step function, not differentiable, difficult to optimize

Squared Loss

$$L(\mathbf{h}(\mathbf{w},\mathbf{x}), \mathbf{y}) = ½*(\mathbf{h}(\mathbf{w},\mathbf{x}) - \mathbf{y})^2$$

- Used in regression
- Differentiable, convex, good candidate for loss function

Logistic Loss

$$L(\mathbf{h(w,x)}, \mathbf{y}) = \log(1+\exp(-\mathbf{y}*\mathbf{h(w,x)}))$$

- Used in artificial neural networks
- Smooth, convex, differentiable

Hinge Loss

$$L(\mathbf{h(w,x), y}) = \max(0, 1 - \mathbf{h(w,x)*y})$$

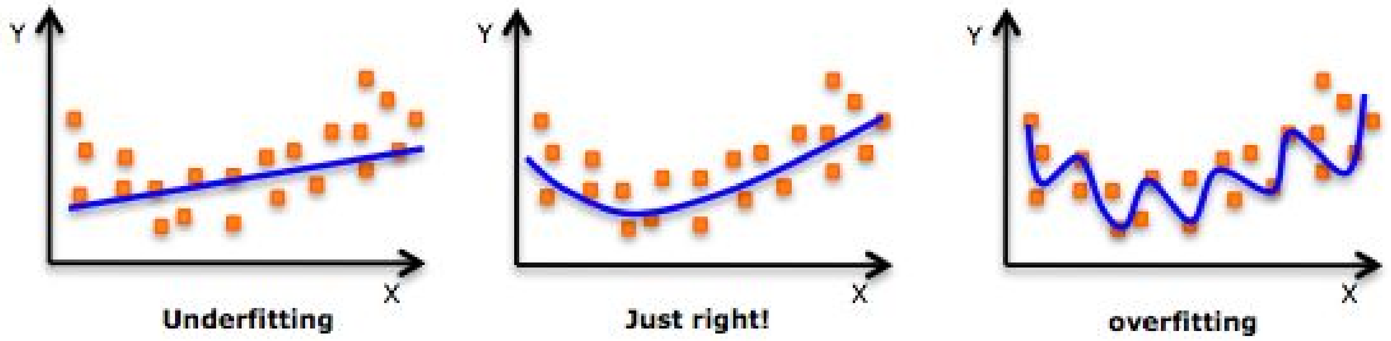
- Used in Support Vector Machines (SVMs)

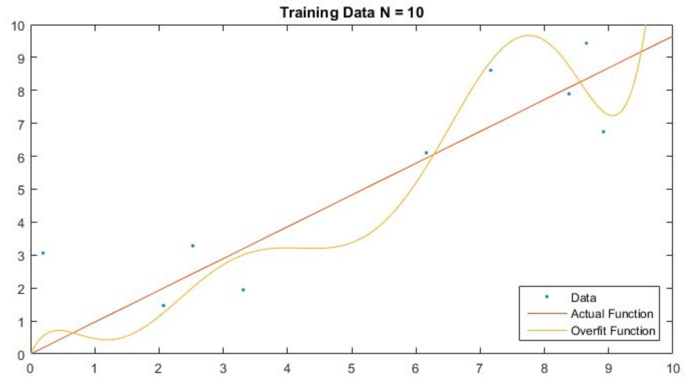
Generalization Error

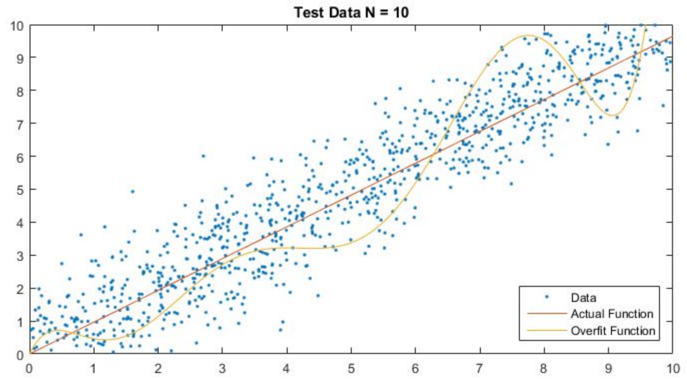
- **Generalization error** - expected error of a classifier on examples not necessarily in the training set. Two components:
  - **bias** - generalization error due to not capturing the full structure of the training set (underfitting)
  - **variance** - generalization error due to the classifier being trained based on patterns specific to the training set (overfitting)
- There is a trade-off here:
  - We don't want the classifier to be so specific to the training set that it can't generalize but we also don't want the classifier to be too general that it can't predict data outside the training set

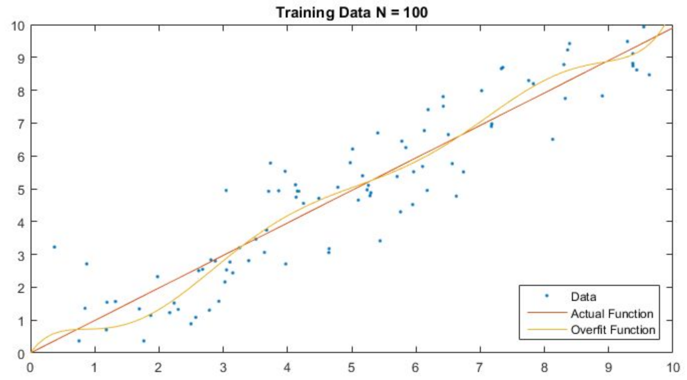
Generalization Error (cont.)

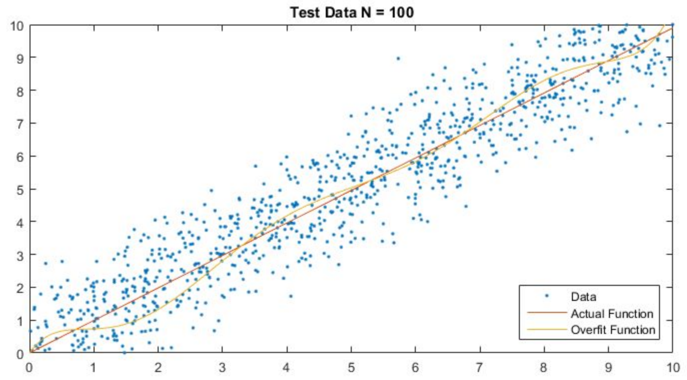
- Given a training set with non-linear structure
    - A linear curve may underfit the data
        - High bias, low variance
    - A non-linear classifier with very high-order terms will overfit the data (e.g. order 10)
        - Low bias, high variance




Training Data N = 10
Data
Actual Function
Overfit Function


Test Data N = 10
Data
Actual Function
Overfit Function


Training Data N = 100
Data
Actual Function
Overfit Function


Test Data N = 100
Data
Actual Function
Overfit Function

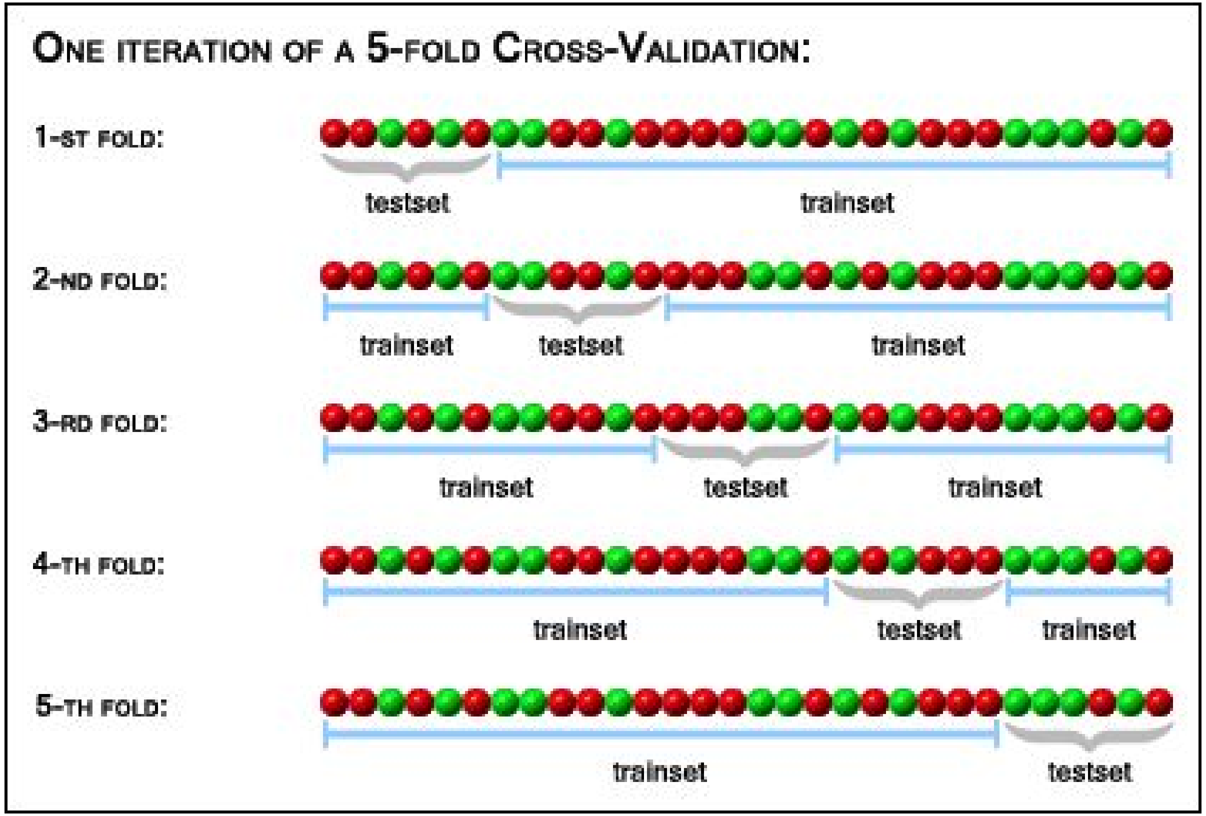
Cross validation

- Let’s say we are given a training set with non-linear structure, but we are not sure what kind of classifier to use
  - Linear or non-linear polynomial with order 2, 3, 4, or 5
  - How do we decide which classifier gives us the best balance between bias and variance?
- Cross validation - do not use the entire training set for training
  - Spit the training set into S_{train} (70% of the data) and S_{test} (30% of the data)
  - Train the different classifiers on S_{train} and choose the one that gives the lowest **loss** on S_{test}

Leave-one-out Cross Validation

- Cross validation wastes 30% of the training set, sometimes training data is very expensive to obtain
- Training set **S** with **m** examples
- Split **S** into **k** disjoint sets of size **m/k**
- For i = 1 to k
  - Leave the ith set out and train each classifer M on the other k-1 sets
  - Calculate the loss function of M on the ith set
- Choose the classifier that has the lowest average **loss** over all sets left out

Leave-one-out Cross Validation



Regularization

- Regularization is another solution to the problem of poor-fitting
- Poor-fitting occurs when we assign too much weight to features based on our training set
- When we minimize a loss function, we are increasing and decreasing weights of the features in order to fit the data optimally
- Regularization prevents dramatic increases and decreases of the weights during the optimization procedure

Regularization (cont.)

- Regularization amounts to adding a term to the minimization problem that is proportional to the feature weights
- Since weights can be positive or negative, it is common to add a norm
- Thus, with regularization we are trying to minimize:

$$\min \left[\sum L(\mathbf{y}, h(\mathbf{w},\mathbf{x})) + \sum \mathbf{w}^2 \right]$$

Intuition Behind Regularization

$$\min \left[\sum L(\mathbf{y}, h(\mathbf{w},\mathbf{x})) + \sum \mathbf{w}^2 \right]$$

- Intuitively, this makes sense:
  - Changing the weights by some amount Δ also adds Δ^2 to the function we are trying to minimize
  - Therefore, we are penalized for giving too much magnitude to feature weights and thus penalized for overfitting

Feature Selection

- Suppose we want to classify a person having cancer risk or not
- There are many, potentially infinite features we could choose from for this classification
  - genomic characteristics
  - male or female
  - occupation
  - smoking or non-smoker
  - socioeconomic status
  - race
- Choosing all the features can be inefficient and moreover we could end up using useless features (i.e. favorite sport’s team)

Feature Selection (cont.)

- First reduce your feature set to size n with basic intuition (exclude useless features)
- With n possible features to choose from, there are 2^n possible subsets of the features to use in our training
- Can’t enumerate over all 2^n subsets (very inefficient)
- Instead, for i=1 to n
    - train your classifier using features 1 to i
    - cross validate to calculate generalization error
    - choose the feature set that results in the lowest generalization error

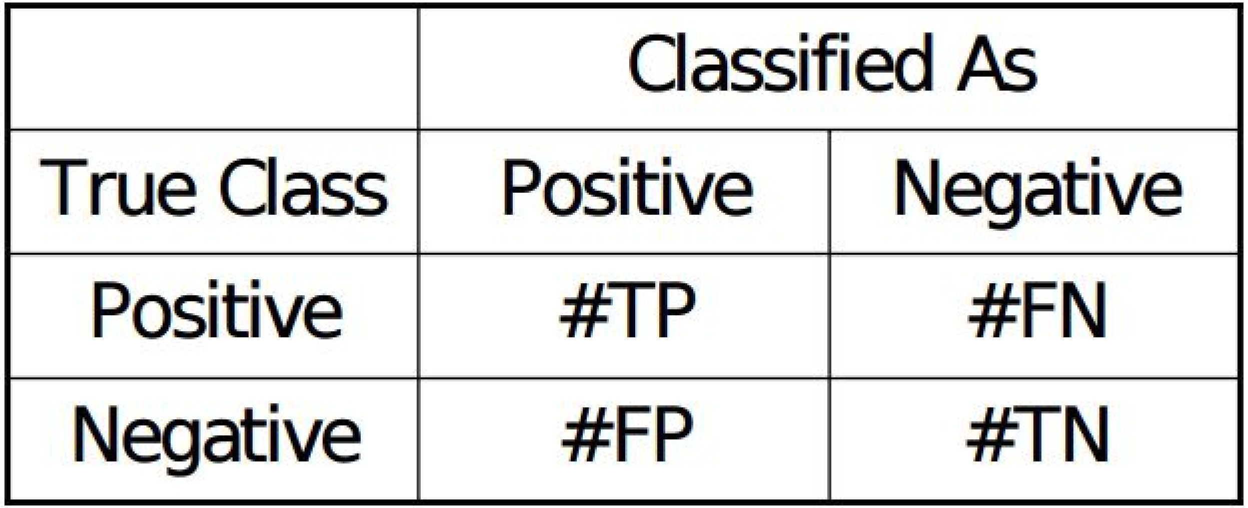
Receiver Operating Characteristic(ROC) Curves

- Calculating the loss function for a classifier does not provide full information about the classifier’s true effectiveness in real-world applications
- A loss function is indifferent to false positives and false negatives, which can have relative importance
- Ex: Getting a false negative on a cancer test can mean death for patients while a false positive will just lead to more testing
    - A loss function assigns equal error to each
- ROC Analysis takes into account false positives and false negatives in deciding how effective a classifier is

Classifier Error Rate

- Success: data is classified correctly
  - True positives (TP) vs. True negatives (TN)
- Error: data classified incorrectly
  - False positives (FP) vs. False negatives (FN)
- Error rate:
  - $e = (FP + FN)/(FP + FN + TP + TN)$

Confusion Matrix

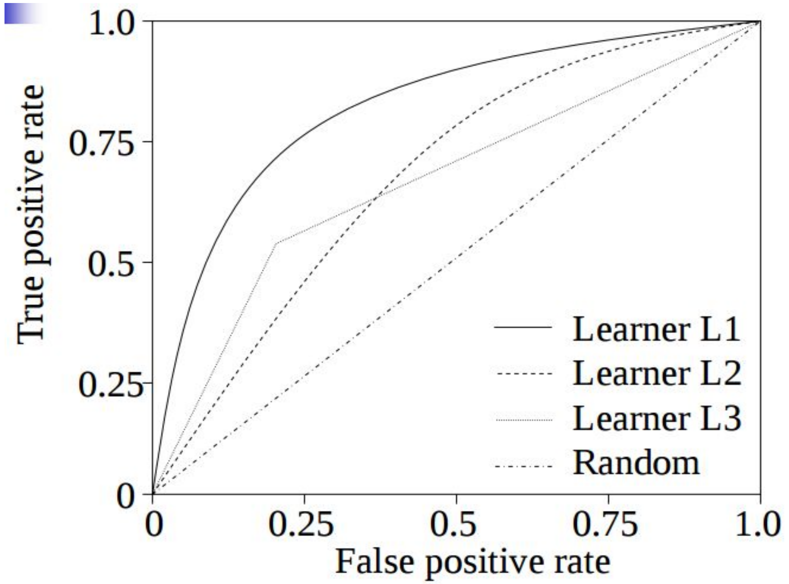


	Classified As	
True Class	Positive	Negative
Positive	#TP	#FN
Negative	#FP	#TN

- True positive rate (Sensitivity) = #TP / (#TP + #FN)
- False positive rate (Specificity) = #FP / (#FP + #TN)

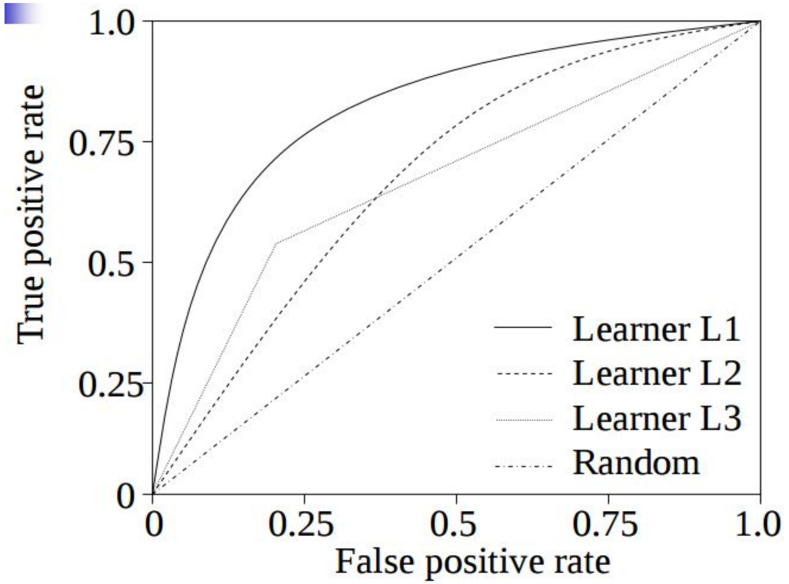
ROC Curve

- Each point on an ROC curve represents a classifier with (FP rate, TP rate) with certain threshold of decision
- For several different threshold values, we can draw a curve for each classifier



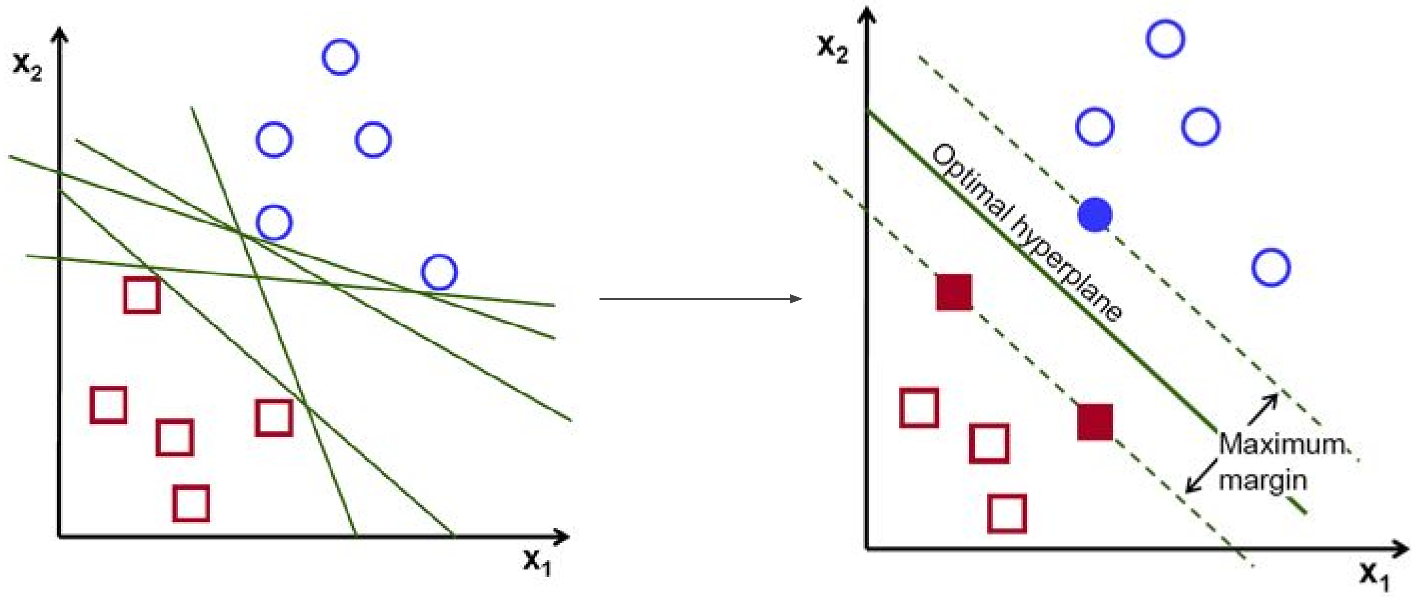
ROC Analysis

- Learner L1 dominates (is better than) all other learners because it’s curve lies above the others
- Learner L2 and Learner L3 have intersecting ROC curves so, in there is a tradeoff between accuracy and cost with deciding between these two



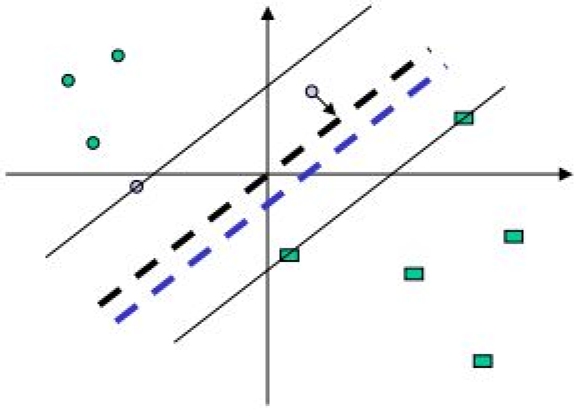
Support Vector Machines (SVMs)

- Classifier that maximizes the margin around a hyperplane that separates the data points into 2 or more sets

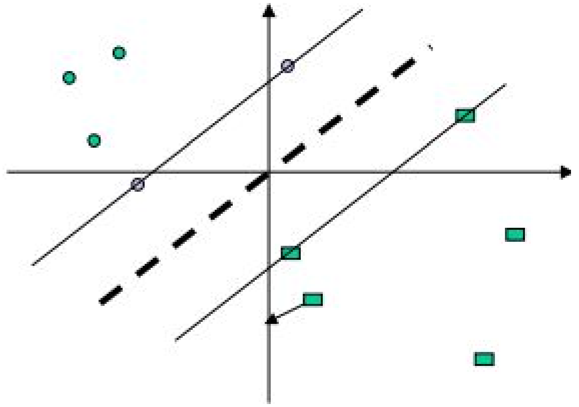


Support Vectors

- The margin only concerns data points closest to the hyperplane
    - These points are called **support vectors**
- Support vectors are the training examples that would move the position of the hyperplane if removed
- We only care about a subset of the training set, the support vectors
    - reason why SVMs are much more efficient than other classifiers that use the entire training set



Moving a support vector moves the decision boundary

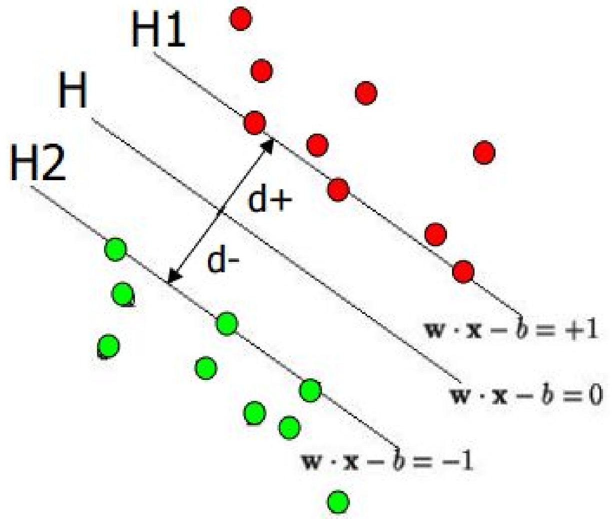


Moving the other vectors has no effect

The algorithm to generate the weights proceeds in such a way that only the support vectors determine the weights and thus the boundary

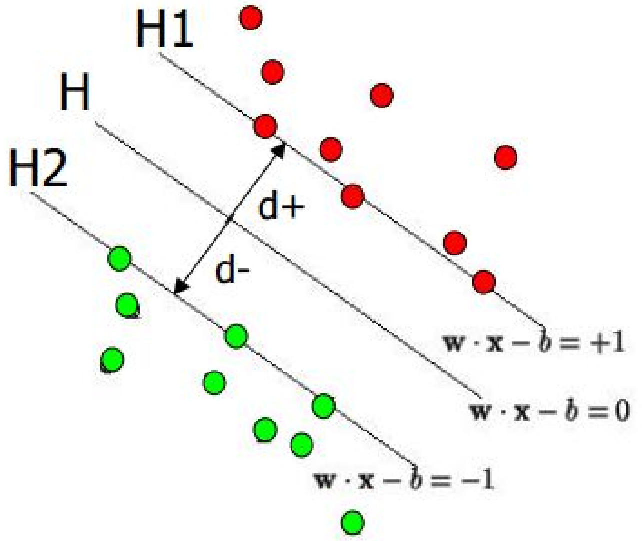
SVM Optimization Objective

- Data points on the hyperplanes H1 and H2 are the support vectors
- We are trying to maximize the margin, or the distance between H1 and H2 such that there are no points between H1 and H2


H1
H
H2
d+
d-
$\mathbf{w}\cdot\mathbf{x}-b=+1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$

SVM Optimization Objective

- Distance between H and H1 is the distance of a support vector from the hyperplane **w·x** - b = 0
- From geometry, the distance from a point to a plane is given by:
  - d = |**w·x**-b| / ||**w**|| = 1 / ||**w**||
  - The margin width is then equal to 2 / ||**w**||
- Optimization goal: maximize the margin width
  - Minimize ||w||
- Optimization constraint:
  - no data points within the margin



H1
H
H2
d+
d-
$\mathbf{w} \cdot \mathbf{x} - b = +1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$

SVM Optimization Objective

In order to maximize the margin, we need to minimize ||w||. With the condition that there are no datapoints between H1 and H2:

$$\left.\begin{array}{l} \mathbf{x}_i \bullet \mathbf{w} + b \geq +1 \text{ when } y_i = +1 \\ \mathbf{x}_i \bullet \mathbf{w} + b \leq -1 \text{ when } y_i = -1 \end{array}\right\}$$

- Can be combined into $\mathbf{y}_i(\mathbf{x}_i * \mathbf{w} + w_0) >= 1$
- What function of **w** do we minimize?
- Least-squares error
  - $\frac{1}{2} * ||\mathbf{w}||^2$
- This is a constrained optimization problem
  - can be solved using Lagrange Multipliers

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta||^2 \text{ subject to } y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i,$$

SVMs Minimize Hinge Loss with Regularization

- The optimization problem we derived using geometry to maximize the margin
- In addition to maximizing the margin, an SVM is also minimizing a loss function with regularization
- In fact, you can derive a “soft-margin” SVM by minimizing the **hinge** loss of the classifier in addition to applying regularization

Weak Learners

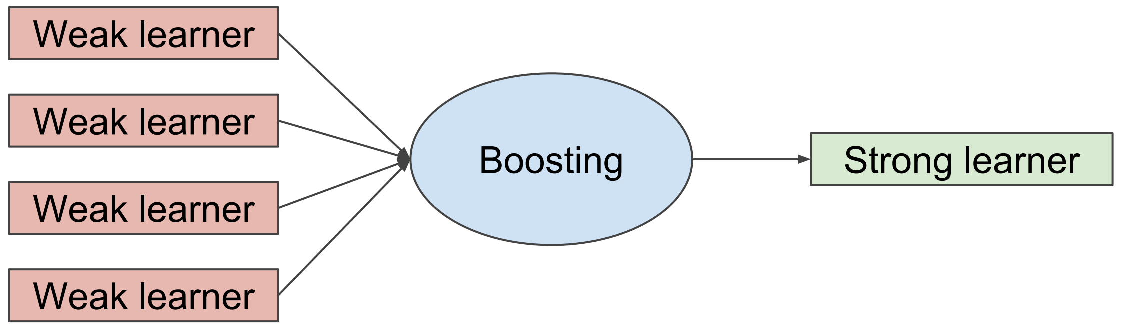
- In binary classification, a weak learner is a classifier that has an error rate of **less than 50%**
- In other words, the weak learners must be better than random guessing
- Some classifiers that can be weak learners:
    - naïve Bayes
    - logistic regression

Weak Learners: Pros and Cons

- Weak learners usually don’t overfit
  - Low variance
- But, they can’t solve hard learning problems
  - High bias
- Can we “combine” many of weak learners into a strong learner?

Combining Multiple Learners

- Idea: combine many weak learners to produce a better overall learner

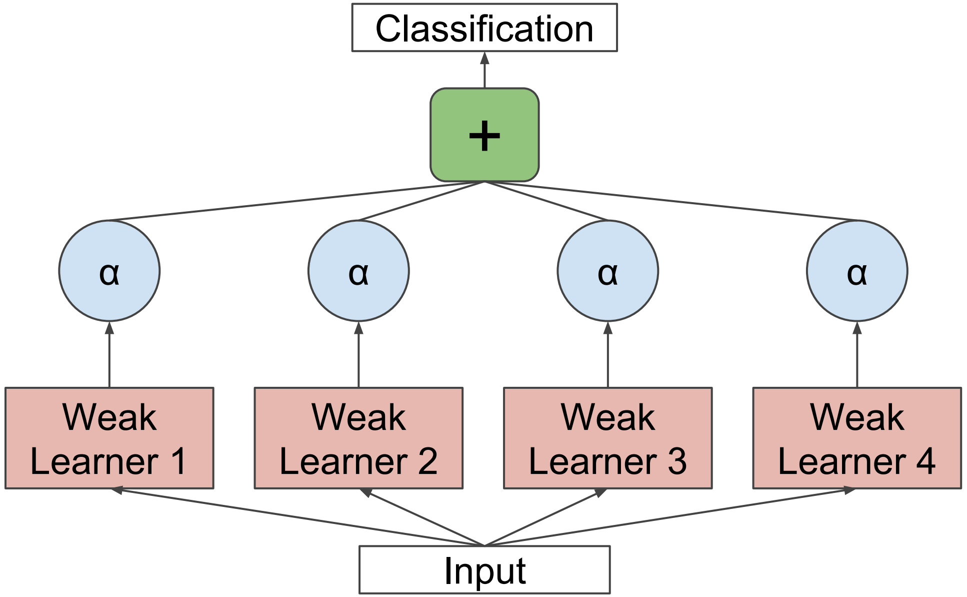


Combining Multiple Learners: Methods

- How do you “combine” many weak learners? A few methods:
- Voting
- Mixture of Experts
- Boosting

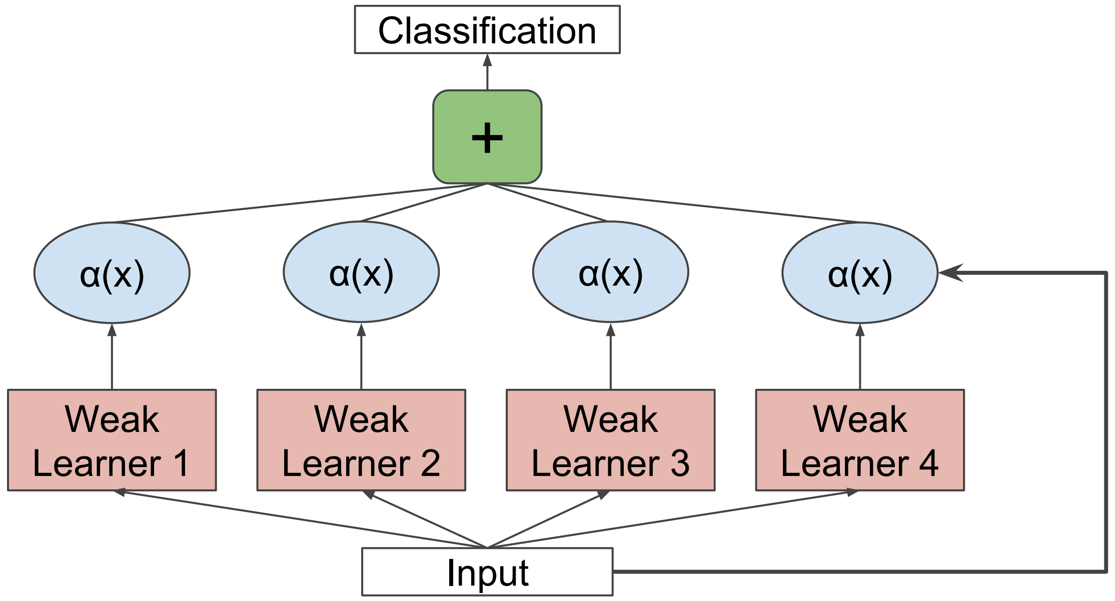
Voting

- We have ***T*** weak classifiers
- Average their predictions with weights (weights are constant)



Mixture of Experts

- Similar to voting, except the weights are not constant



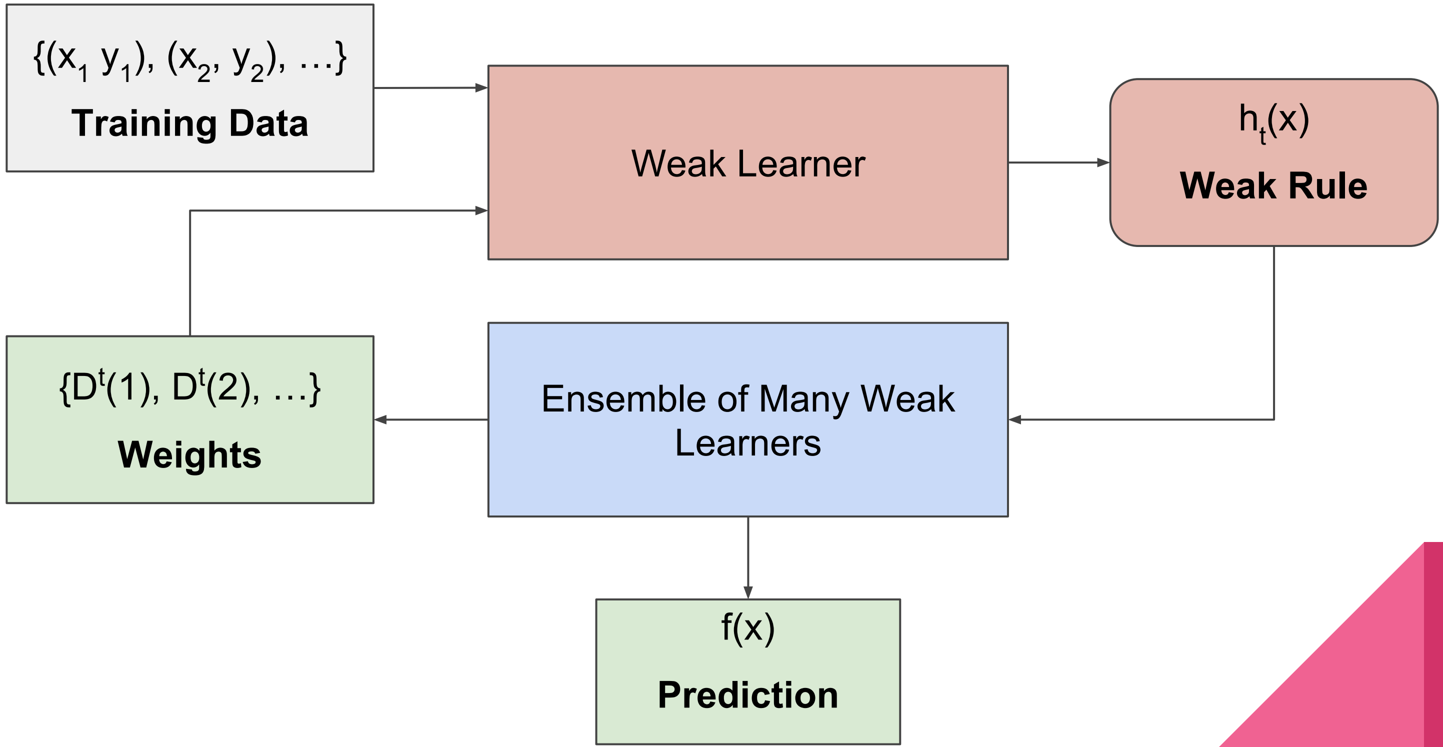
Boosting

- In both voting and mixture of experts, the weak learners are “static” with respect to the inputs
- What if you retrain classifiers to learn about different parts of the input space?
- Called **boosting**

Boosting

- Convert your training data to “weighted” data
  - **[x(i), y(i)] ⇒ [D^T(i) x(i), y(i)]**
  - **D^T(i)** is the “weight” of the **i**th data point at time **T**
- Intuition: some data points matter more than others
- Retrain your weak learners at each training step based on the mistakes of previous learners

Boosting



$\{(x_1\ y_1), (x_2, y_2), \ldots\}$
Training Data
Weak Learner
$h_t(x)$
Weak Rule
$\{D^t(1), D^t(2), \ldots\}$
Weights
Ensemble of Many Weak Learners
$f(x)$
Prediction

Boosting

- Many different weighting schemes
- Adaboost (adaptive boosting) is one of the most popular
- Adaboost is parallelizable and distributable
  - GPU implementations as well as MapReduce implementations

Boosting Pros and Cons

- Pros:
  - Relatively few parameters to tune (number of rounds, a constant α_t)
  - Fast and can be parallelized
  - Theoretical guarantees on training error
- Cons:
  - Performance depends heavily on the type of weak learners
  - Susceptible to noise
    - Possible alternatives: BrownBoost, Gentle Adaboost