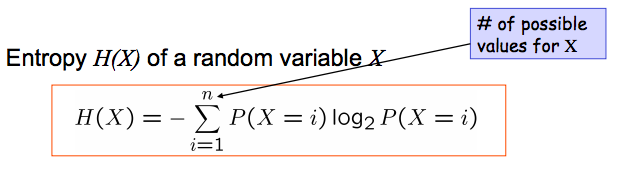
* 01—Introduction
  + Machine learning is the study of algorithms that improves their performance P at some task T with some experience E
  + Types of Learning
    - 1) Supervised (inductive) learning
      * given: training data + desired outputs
    - 2) Unsupervised learning
      * given: training data (without desired outputs)
    - 3) Semi-supervised learning
      * given: training data + a few desired outputs
    - 4) Reinforcement learning
      * rewards from sequence of actions
  + If training and test examples are NOT independent, requires *collective classification*. If test distribution is different, requires *transfer learning*
  + Every ML algorithm has three components: 1) representation, 2) optimization, 3) evaluation
  + 1980s: advanced decision tree and rule learning, 2000s: SVM and kernel methods
* 02—Decision Trees
  + each instance *x* in *X* is a feature vector
  + Performance bias: Ockham’s Razor—entities are not to be multiplied beyond necessity (Idea: the simplest explanation is the best)
  + Choosing the best attribute
    - Least-values: choose attribute with fewest values, most-values: choose attribute with most values
    - ID3 algorithm uses the Max-Gain method of selecting the best attribute
    - Information Gain—measures the level of impurity in a group of examples
    - Entropy is a common way to measure impurity
      * H(X) = 
      * Entropy of a group with 50% in either class = 1, = 0 in example for which all belong to same class
    - Information Gain = **Entropy(Parent) – Average Entropy(Children)** = I(X,Y) = H(X) – H(X|Y) = H(Y) – H(Y|X)
    - Disadvantage of Information Gain
      * Prefers attributes with large numbers of values that split that data into small, pure subsets
        + Quinlan’s gain ratio uses normalization to improve this

GainRatio(X,A) = Gain(X,A) / SplitInfo(X,A)

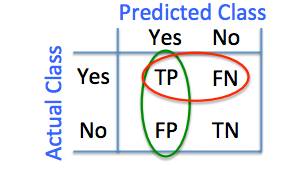
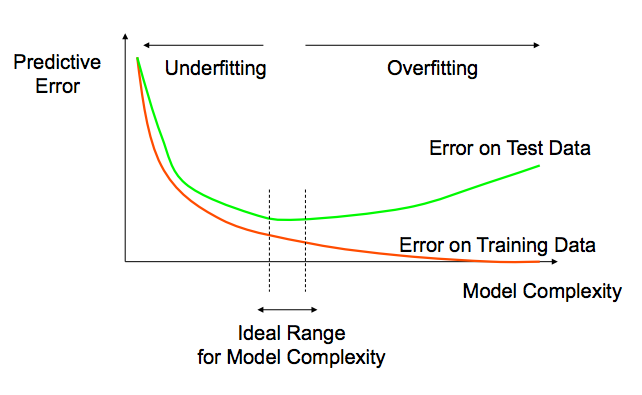
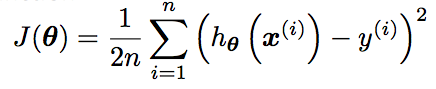
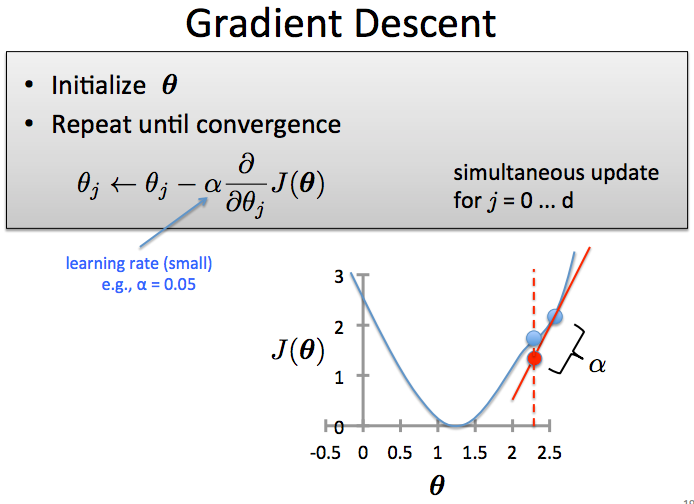
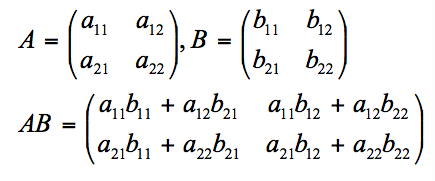
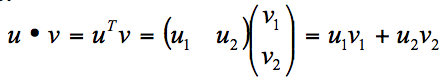
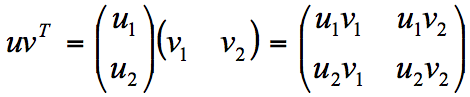
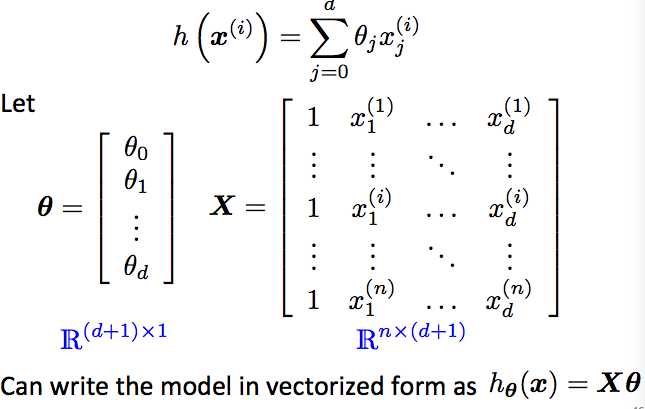
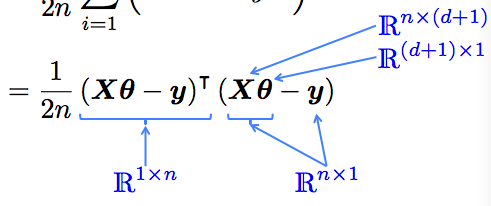
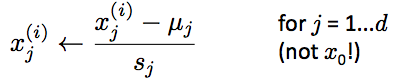
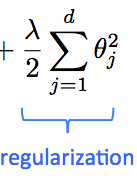
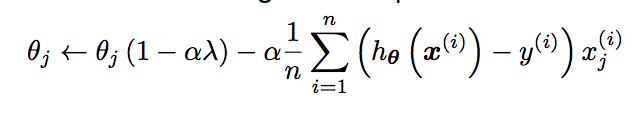
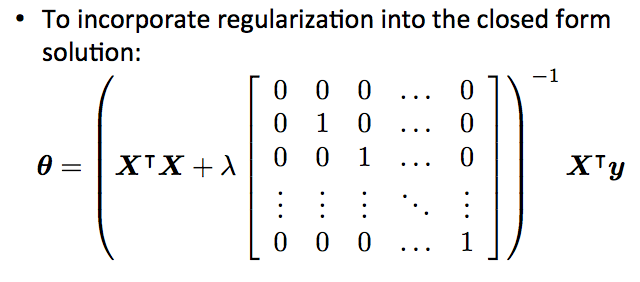
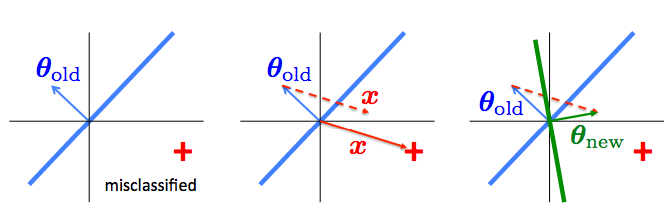
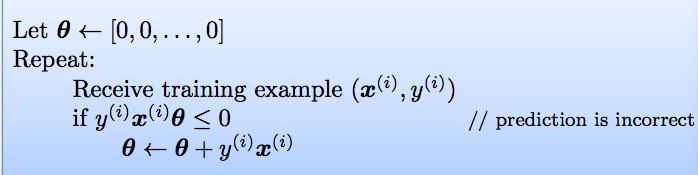
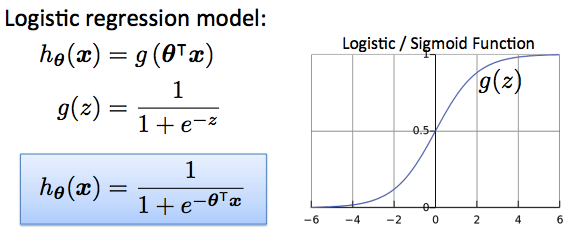
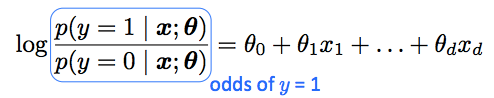
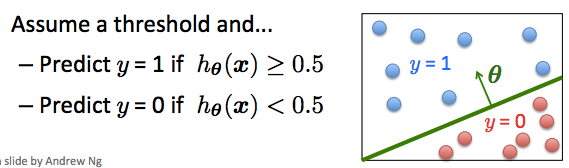
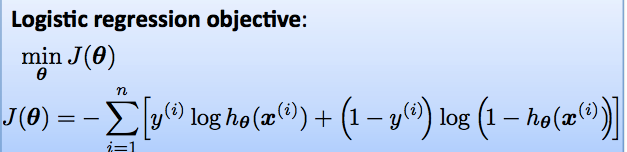
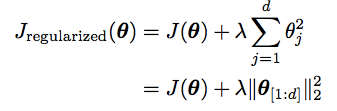
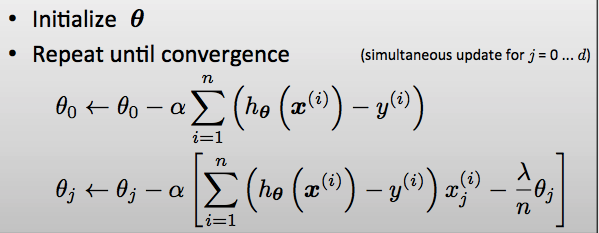
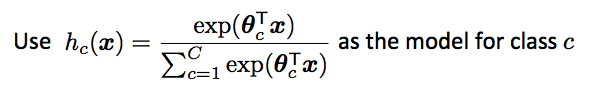
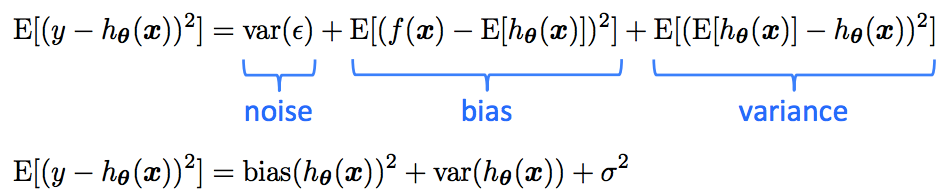
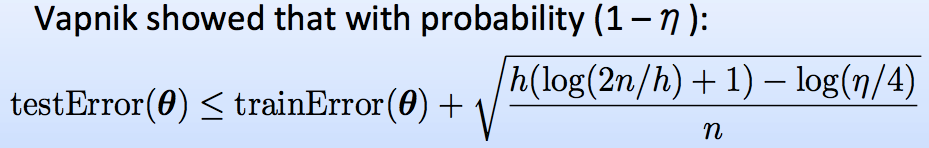
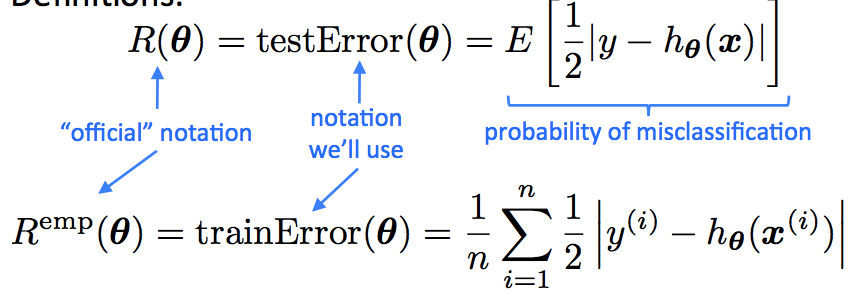
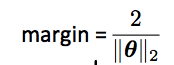
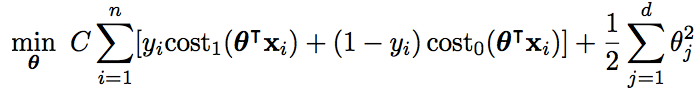
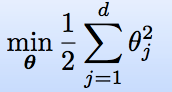
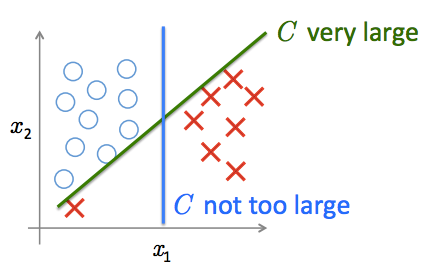
SplitInfo(X,A) = -((1/6)log(1/6) + (1/3)log(1/3) + (1/2)log(1/2))…

* + - Noisy Data
      * 1) two examples have same attribute/value pairs, but different classifications
      * 2) some values of attributes are incorrect because of errors in the data acquisition process of the preprocessing phase
    - Overfitting…how to avoid
      * Stop growing when data split is not statistically significant
      * Acquire more training data
      * Remove irrelevant attributes
      * Grow full tree, then post-prune
      * Reduced-Error Pruning
        + Split data into training and validation sets…grow tree based on training set and greedily remove the node that most improves the validation set accuracy until further pruning is harmful
        + C4.5:

1) Convert tree to equivalent set of rules

2) prune each rule independently of others

3) sort final rules into desired sequence for use

* + - Models Summary…
      * (<= hundreds of thousands) ID3—use max-gain to select best attribute
      * (<= hundreds of thousands) C4.5—prune each rule independently and then sort rules
      * (<= millions) SPRING, SLIQ—multiple sequential scans of data
      * (<= billions) VFDT—at most one sequential scan
    - Summary of Decision Trees
      * Bad for extracting linear combinations of features, predictive power, sacrifices predictive power, requires fixed-length feature vectors. Good for handling data of mixed type, missing values, computational scalability, dealing with irrelevant inputs, fast and simple, converting to rules, with noisy data.
* 03—K-Nearest Neighbor
  + A type of instance-based/memory-based learning
  + 4 aspects of an instance-based learner:
    - 1) distance metric 2) how many nearby neighbors to look at 3) a weighting function (optional) 4) how to fit with the local points
  + Pros: good job of noise smoothing Cons: lack of gradients, jerkiness, can lose detail that 1-nearest neighbor would give
* 04—Batch Machine Learning
  + Classification Metrics:
    - Accuracy = (# correct predictions) / (# test instances)
    - Error = 1 – accuracy = (# incorrect predictions) / (# test instances)
  + 
  + accuracy = (TP + TN) / (P + N)
  + precision = (TP) / (TP + FP) 🡪 P(randomly selected event is relevant)
  + recall = TP / (TP + FN) 🡪 P(randomly selected relevant document is retrieved)
  + 
  + k-fold cross-validation (remember from the homework)
  + Multiple trials of k-fold CV
* 04—Linear Regression
  + Cost function: 
  + 
  + 
* Linear Algebra
  + (Ax)^T = x^TA^T
  + 
  + 
  + 
  + h(x) = (theta)^T\*x
  + h\_theta(x) = X\*theta (when the model has *n* instances)
  + 
  + Cost = J(theta) = 
  + Closed form solution… => 
  + Gradient vs. Closed Form
    - Gradient can support incremental learning, works well when *n* is large
    - Closed Form is slow if *n* is large but no need for *a*
  + Improving Learning
    - Feature scaling—ensure that features have similar scales to make gradient descent converge must faster
    - Feature standardization—rescales features to have zero mean and unit variance
      * Replace each value with…
      * Must apply same transformation for training AND prediction
      * Outliers can cause problems
  + Underfitting🡪 high bias, overfitting 🡪 high variance
  + Regularization—controlling the complexity of the learned hypothesis
    - 
    - Gradient step can be rewritten as 
    - 
* 04—Linear Classification
  + 
  + 
  + Online Perceptron Learning Algorithm:
    - 
  + Online learning—the learning mode where the model update is performed each time a single observation is received
  + Batch learning—the learning mode where the model update is performed after observing the entire training set
  + Improving the perceptron
    - Use a combination of multiple perceptrons
    - Use the intermediate theta’s (voted or averaged)
* 05—Logistic Regression
  + Classification based on probability…instead of just predicting the class, give the probability of the instance being that class
  + Logistic regression
    - 
    - z = theta^T\*x
    - h\_theta(x) = estimated p(y = 1|x;theta)
  + 
  + 
  + Can’t use squared loss as in linear regression, because using the logistic regression model results in a non-convex optimization
  + 
  + Can regularize the same way as with linear regression…
    - 
  + 
  + 
* 06—Learning Theory
  + if d dimensions, |X| = 2^d, |h| = 2^|X|
  + 
  + Bias-variance…high regularization = high bias = low variance
  + Training error drives down bias but ignores variance
  + Idea to get a guarantee of the following form:
    - testingError <= trainingError + f(n,h,p) where n = size of training set, h = measure of the model complexity, p = probability that this bound fails
    - 
  + VC dimension—the number of data points a model can learn perfectly for all possible assignments of labels
    - = maximum number of points that can be arranged so that the class of models can shatter
    - For hyperplanes, VC dimension = *d* + 1
  + 
  + Takeaway—if |*H*| is very big relative to |*n*|, then models with low training error are lucky. But, if |*H*| is constrained and/or size of training data set *n* is large, then low training error is likely to be evidence of low generalization error
* 07—SVMs and Kernels
  + Why might predictions be wrong? 1) Partial observability (hard or soft) 2) noise in the observation x (measurement error, instrument limitations)
  + SVMs
    - Goal is to maximize the margin
      * 
    - 
    - 
  + 
  + p = 1 / ||theta||2
  + Kernel Methods…Making the non-linear linear
  + 
  + SVMs vs. Logistic Regression
    - n = # training examples, d = # features
    - if d is large relative to n
      * use logistic regression or SVM with linear kernel
    - if d is small, n is intermediate
      * use SVM with Gaussian kernel
    - if d is small, n is large
      * Create/add more features, then use logistic regression or SVM without a kernel
    - SVMs are slow to train/predict for huge data sets, relatively speaking
* 08—Ensemble Learning
  + Successful ensembles require diversity­
    - If overfitting, vary the training sets. If features are noisy, vary the set of input features.
  + Manipulating the Training Data
    - Bootstrap replication—given n training examples, construct a new training set by sampling n instances with replacement
    - Bagging—create bootstrap replicates of training set, and train a classifiers for each replicate and then average the output of all classifiers
    - Boosting—creates an ensemble of weak hypotheses by repeatedly emphasizing mispredicted instances
      * Final model is a weighted combination of members
      * AdaBoost
        + Works best with “weak” learners, typically high bias classifiers. Works even when weak learner has an error rate just slightly under 0.5
        + Does not explicitly regularize the model
        + Does NOT overfit, contradicting VC theory

Drives down test error even after the training error reaches zero

* + - * + Pros

Fast and simple to program, no parameters to tune, no assumptions on weak learner

* + - * + Cons

Can fail with insufficient data, overly complex weak hypotheses, susceptible to noise

* + - * + **“AdaBoost with trees is the best off-the-shelf classifier in the world”**
    - Random Forests—construct decision trees on bootstrap replicas, restricting the node decisions to a small subset of features for each node. Average the output of all trees.