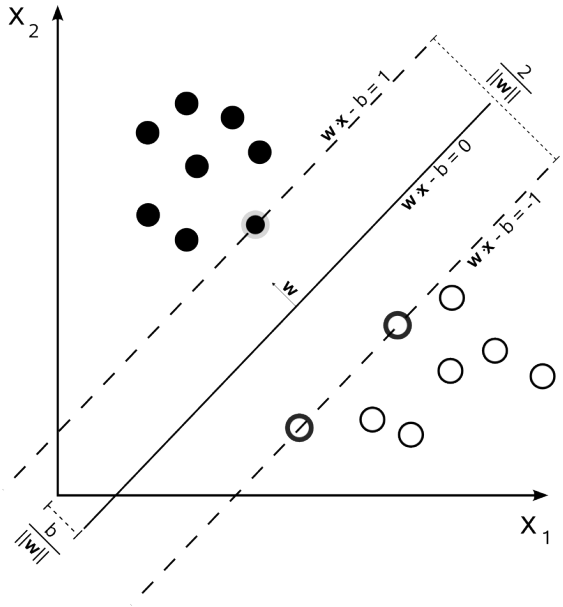
Computer Science 577 Notes
Machine Learning

Mendel C. Mayr

February 8, 2015



Contents

1 Decision Tree Learning

1.1 Information Gain

Information gain is used to determine the (feature to) split
Information theory: entropy and information gain:

(i) Entropy $= H(X) = -\sum_{x \in X} P(X = x) \log_2 P(X = x)$

(ii) Conditional entropy $= H(Y|X) = -\sum_{x \in X} P(Y|X = x)$, where
$H(Y|X = x) = \sum_{y\ inY} P(Y = y|X = x) \log_2 P(Y = y|X = x)$

(iii) Mutual information (information gain) $= I(X, Y) = H(Y) - H(Y|X)$

Alternative metric: Gini coefficient, i.e. product of probabilities for each of (2) outcomes for the feature

Limitation of information gain: biased toward tests with many outcomes (i.e. features with many possible values)
C4.5 uses gain ratio: $SplitInfo(D, S) = -\sum_{k \in S} |D_k|/|D| \log_2(D_k/D)$, and $GainRatio = I(D, S)/SplitInfo(D, S)$

1.2 Decision Tree Algorithms

Overfitting: $h \in H$ overfits the training data D if there is an alternative model $h' \in H$ such that $error(h) > error(h')$ yet $error_D(h) < error_D(h)$
Avoiding overfitting in decision tree learning:

(i) Early stopping: stop if further splitting not justified by statistical test

(ii) Post-pruning: grow large tree, then prune nodes using tuning set
Iteratively eliminate nodes until further reductions reduce accuracy

Lookahead: instead of evaluating using information gain, look ahead to see what splits at the next level would be, and measure information gain at a deeper level

Continuous features: use threshold-based boolean attribute, treshold determined by sorting examples according to the featurem and generating candidate thresholds between adjacent examples with different class values
Threshold chosen from candidates based on information gain

Training examples with missing attribute values: possible strategies

(i) At node n, upon encountering a missing feature value, assign it the value most common among examples at node n, or most common among examples at the node n that also have the same class value

(ii) Assign fractional value to attribute for the example and pass fractional example to children for purposes of evaluating information gain

2 Instance-Based Learning

2.1 K-nearest Neighbor

k-nearest neighbor classification: given an instance x_q to classify, find k training-set instances that are most simimlar or x_q
Return the class value: $\hat{y} = argmax_{v \in values(Y)} \sum_{i=1}^{k} \delta(v, y_i)$
Various determinations of distance:

(i) Hamming distance: number of features with differing values

(ii) Euclidean distance: $\delta(x_i, x_j) = \sqrt{\sum_f (x_{if} - x_{jf})^2}$

(iii) Manhattan distance: $\delta(x_i, x_j) = \sum_f (x_{if} - x_{jf})^2$

k-nearest neighbor regression: given an instance x_q, find the k nearest training-set instances and return $\sum_{i=1}^{k} \delta(v, y_i)$
Distance-weighted nearest neighbor: instances contribute to prediction according to their distance from x_q

k-nearest neighbor does almost nothing at training time, and offsets costs to classification/prediction time
Strategies to speed up k-nearest neighbor

(i) Don't retain every training instance: edited nearest neighbor
Select subset of instances that still provide accurate classifications:

(a) Incremental deletion: delete from memory all training instances redundant to classification
(b) Incremental growth: if training instances insufficient to classify training instance, add to memory

(ii) Use data structure to look up nearest neighbors (k-d tree)

k-d trees (A^* instance search): each node stores one instance, and splits on the median value of the feature having the highest variance
Nodes are pushed to the A^* priority queue with the value indicating the minimum possible distance to the query based on the threshold for the split

Strenghts of instance based learning: simple, efficeint training, easily adapts to on-line nearning, rubust to noisy data, etc.
Limitations: sensitive to range of feature values, senstivie to irrelevant and correlated features, inefficient classification, no insight into problem domain (i.e. lacks modeling of problem)

2.2 Linear Locally Weighted Regression

Locally weighted linear regression: $f(x) = w_0 + w_1 a_1(x) + \ ... \ + w_n a_n(x)$
Local approximations to have query point fit local training examples:

1. Minimize squared error over just k nearest neighbors:
   $E_1(x_q) = (1/2) \sum_{x \in k \text{ nearest neighbors of } x_q} (f(x) - \hat{f}(x))^2$
2. Minimize squared error over entire set of D of training examples, using decreasing function K:
   $E_2(x_q) = (1/2) \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$
3. Combination of the previous: let N be k nearest neighbors of x_q

$$\frac{1}{2} \sum_{x \in N} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

3 Probability and Bayesian Learning

Note: for more rigourous details, see Heckerman's Bayesian Network Learning Tutorial

3.1 Probabilistic Machine Learning Concepts

Recall Bayes theorem: $P(A|B) = P(B|A)P(A)/P(B)$

Brute-force MAP learning algorithm:

(i) Given information D, for each hypothesis $h \in H$, calculate the posterior probability $P(h|D) = P(D|h)P(h)/P(D)$

(ii) Output hypothesis h_{MAP} with highest posterior probability $h_{MAP} = argmax_{h \in H} P(h|D)$

Other probabalisitic concepts in machine learning:

(i) Maximum likelihood and least-squared error hypotheses

(ii) Maximum likelihood hypotheses for predicting probabilities

(iii) Minimum description length principle

(iv) Bayes optimal classification: $argmax_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$

(v) Gibb's Algorithm: choose hypothesis $h \in H$ at random, according to posterior probaiblity distribution over H, and use h to predict the classification of the next instance x

3.2 Bayesian networks

A Bayesian network consists of a directed acyclic graph and a set of conditional probability distributions
In the each Directed Acyclic Graph (DAG):

(i) Each node denotes a random variable

(ii) An edge from X to Y represents that X directly influences Y

(iii) Each node X contains a conditional probability distribution representing $P(X|Parents(X))$

(iv) Each variable X is independent of its non-descendants given its parents

(v) Each variable X is independent of all others given its Markov blanket

Advantages of Bayesian network representation:

(i) Captures indepdendence and conditional indpendence where they exist

(ii) Encodes the relevant portion fo the full joint distribution

(iii) Graphical representation gives insight into complexity of inference

Inference task: given values for some variables in the network (evidence) and set of query variables, compute posterior distribution over query variables
Hidden variables: neither evidence nor the query variables
Baysean networks allow for any set to be evidence and any set to be query
Inference by enumeration: consider the chain rule $P(x_1, ..., x_n) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)...P(x_n|x_{n-1}, ..., x_1)$
Posterior probability on query variables can be found via independence, chain rule, and marginalization

Parameter learning task: given set of training instances and graph structure, infer parameter of conditional probability distributions
Strcture learning task: given set of training instances, infer graph structure (and possibly parameters of CPDs)
For parameter learning, use maximum a posteriori (MAP) estimation: e.g. m-estimates $P(X = x) = (n_x + p_x m)/(\sum_{v \in Values(X)} n_v) + m$, where p_x is prior probability of value x, m is number of virtual instances, and n_v is number of occurences of value v

3.3 Expectation Maximization

Missing data (hidden variables, values missing at random): values can be imputed using Expectation Maximization (EM)
Iterate until convergence:

(i) Expectation step: using current model, compute expectation over missing values (missing temporarily take expected values)

(ii) Maximization step: update model parameters with those that maximize probability of data (MLE or MAP)

Note that k-means unsupervised clustering is a form of expectation maximization
Expectation maximation can be hard (takes most likely value) or soft (expectation is probabiltiy distribution)

Expectation maximation for parameter learning:

(i) Expecation step: compute probability of each completion of incomplete data points, i.e. answering query over missing variables given others

(ii) Maximization step: use completed data set to update Dirchlet distributions, except counts can be fractional, update conditional probability tables

Subtelty for parameter learning: overcounting based on number of iterations required to converge to settings for missing values. After each expectation step, reset Dirichlet distributions before repeating maximization step

Problems with expectation maximization: only finds local optimum, deterministic with respect to priors

3.4 Learning Network Structure

Chow-Liu algorithm: learns tree structure that maximizes likelihood of training data

(i) Compute weight $I(X_i, X_j)$ of each possible edge (X_i, X_j)

(ii) Find maximum-weight spanning tree: use mutual information to calculate
$I(X, Y) = \sum_{x \in values X} \sum_{y \in values Y} P(x, y) \log_2 P(x, y)/(P(X)P(Y))$

Prim's algorithm (given (V, E)):

(a) $V_{new} = \{v\}$ where $v \in V$ (arbitrary)

(b) $E_{new} = \{\}$

(c) Repeat until $V_{new} = V$: choose edge (u, v) in E with max weight where u is in V_{new} and v is not. Add v to V_{new} and (u, v) to E_{new}

(d) Return $(V_{new}, E_{new}$, a maximum spanning tree

Kruskal's algorithm (given (V, E)):

(i) $E_{new} = \{\}$

(ii) For each (u, v) in E ordered by weight (high to low):
Pop (u, v) from E and add to E_{new} if it does not create a cycle

(iii) Return V and E_{new}, a maximum spanning tree

(iii) Assign edge directions in maximimum-weight spanning tree
Pick a node for the root, assign edge direction

Heuristic search for structure learning: each state in search space represents Bayes net structure
Search approach requires specification of

1. Scoring function: $score(G, D) = \sum_i score(X_i, Parents(X_i) : D)$
Thus, a network can be scored by summing terms over nodes, and changes can be efficently scored via a local search
2. State transition operators: adding an edge, deleting in edge, reversing an edge
3. Search algorithm: hill climbing or sparse candidate search

Bayesian network hill climibing search: greedy algorithm
Bayesian network sparse candidate search (given data set D, initial network B_0, parameter k):

(i) Let $i = 0$

(ii) Repeat until convergence:

(a) Increment i

(b) Restrict step: select for each variable X_j a set $C_j^i(|C_k^i|)$ of candidate parents

(c) Maximize step: find network B_i maximizing score among networks where $\forall X_j, Parents(X_j) \subset C_j^i$

(iii) Return B_i

Restriction step in sparse candidate search:
Fpr the first iteration: candidate parents computed using mutual information
$I(X, Y) = \sum_{x,y} P(x, y) \log (P(x, y)/(P(x)P(y)))$
Kullback-Liebler divergence: distance measure between two distributions P and Q

$$D_{KL} = (P(X)||Q(X)) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

KL-divergence assesses discrepancy between network's estimate $P_{net}(X, Y)$ and empirical estimate $M(X, Y) = D_{KL}(P(X, Y)||P_{net}(X, Y))$

Algorithm for restriction step in sparse candidate (given data D, current network B_i, parameter k):

(i) For each variable X_j

(a) Calculate $M(X_j, X_i)$ for all $X_j \neq X_i$ such that $X_l \notin Parents(X_j)$

(b) Choose highest ranking $X_l ... X_{k-s}$ where $s = |Parents(X_j)|$

(c) Include current parents in candidate set to ensure monotonic score improvements
$C_j^i = Parents(X_j) \cup X_l ... X_{k-s}$

(ii) Return C_j^i for all X_j

Scoring function for structure learning: maxmize data probability, but penalize complexity
General approach: $argmax_{G,\theta_G} \log P(D|G, \theta_G) - f(n)|\theta_G|$
Akaike information criterion (AIC) $f(n) = 1$, Bayesian Information Criterion (BIC) $f(n) = \log (n)/2$

3.5 Naive Bayes and Tree Augmented Network

Naive Bayes assumption: all features X_i are conditionally independent given class Y
$P(X_1, ..., X_n, Y) = P(Y)\Pi_{i=1}^{n} P(X_i|Y)$
Unaugmented tree starts with edge from node Y to each feature $X_1, ..., X_n$
Learning: estimate $P(Y = y)$ for each value of Y, estimate $P(X_i = x|Y = y)$ for each X_i
Classification done using Bayes rule:

$$P(Y = y|X) = \frac{P(y)P(X|y)}{\sum_{y' \in values(Y)} P(y')P(X|y')} = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{\sum_{y'=values(Y)} (P(y')\prod_{i=1}^{n} P(x_i|y'))}$$

Tree Augmented Network (TAN) algorithm: learns tree structure to augment edges of naive Bayes network

(i) Compute weight $I(X_i, X_j|Y)$ for each possible edge (X_i, X_j) between features

(ii) Find maximum weight spanning tree (MST) for graph over $X_l ... X_n$

(iii) Assign edge direction in MST

(iv) Construct a TAN model by adding node for Y and an edge for Y to each X_i

4 Machine Learning Methdology

4.1 Partitioning of Data

Evaluation of learning models: given instance set X and probability distribution D defining the probaiblity of encountering an $x \in X$, learner must learn target concept (function) $f \in H$
Evaluation methdology seeks to answer two questions: what is the best accuracy for a hypothesis h, learned from n instances, applied to future instances drawn according to D, and what is the probably error in this accuracy estimate

Distinction between sample error and true error:

(i) Sample error: error of h with respect ot target function f and data sample S
$error_S(h) = (1/n) \sum_{x \in S} \delta(f(x), h(x))$, where $\delta(a, b) = 1$ if $a \neq b$, 0 otherwise

(ii) True error: $error_D(h) = Pr_{x \in D}[f(x) \neq h(x)]$

Limitations of singular training/test partitions:

(i) Not enough data for sufficiently large training and test sets

(ii) Single training set does not show how sensitive accuracy is to particular training sample

Random resampling: repeatedly partitioning data into training and test sets
Stratified sampling: statify instances by class and select (i.e. maintain class proportions)
Cross validation: partition data into n subsamples. Iteratively test on one subset, train on rest
Leave-one-out cross validation: n is the number of instances

4.2 Performance Evaluation

Receiver Operating Characteristic (ROC) curve: true positive-rate vs false positive-rate as threshold as confidence of an instance being positive is varied
Algorithm for creating an ROC curve:

(i) Sort test-set prediction according to confidence that each instance is positive

(ii) Step through sorted list from high to low confidence

  (a) Locate threshold between instances with opposite classes

  (b) Compute TPR, FPR for instances above threshold

  (c) Output (FPR, TPR) coordinate

Points plotted on ROC curve can be interpolated to form convex hull
Having a high TPR or low FPR does not indicate accuracy in unbalanced data sets

Alternative accuracy metric: recall and precision

(i) Recall (TP rate) = TP/(actual positives) = TP/(TP + FN)

(ii) Precision = TP/(predicted positives) = TP/(TP + FP)

Precision/recall curve: plots prevision vs. recall as threshold as confidence of an instance being positive is varied

Single ROC/PR curve with cross-validation: multiple approaches

(i) Assume confidence values comparable across folds
Pool predicitons from all test sets and plot curve from pooled predictions

(ii) Plot individual curves for all test sets, viewing each as function
Plot average curve for set of functions

ROC and PR curves allow predictive performance to be assessed at various levels of confidence, assume binary classification, and can be summarized by the integral
ROC curves: insensitive to class distribution changes, can identify optimal classification thresholds for tasks with differential misclassification costs
PR curves: show fraction of predictions that are false positives, suited for tasks with many negative instances

4.3 Confidence Intervals and Learning Tasks

Avoiding pitfalls: questions when applying learning tasks

(i) Held-aside test data should be representative of collecting new data

(ii) Folds of cross validation should only use training data for the fold
At no point in preprocessing should test case labels be accessed

(iii) Repeated modificiations of learning algorithm or preprocessing could lead to overfitting

Confidence intervals on error: suppose we have learned model h, test set S containing n instances drawn independently of each other and independent of h, with $n \geq 30$, and h makes r errors over the n instances
Estimate of error is $error_S(h) = r/n$
With approximately N probability, the true error is in the interval

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

where z_N is a constant that depends on N, e.g. $z_{0.95} = 1.96$
Confidence interval follows from normal approximation of binomial distribution
An $N\%$ confidence interval on parameter p is an interal that is expected with $N\%$ probability to contain p
In using confidence intervals, consider (central limit theorem) that a large number independent, identically distribution random variables approximately follows a normal distribution

Difference in error of two hypotheses: $\hat{d} = error_S(h_1) - error_S(h_2)$
$N\%$ confidence interval for d is:

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}}$$

4.4 Comparing Learning Models and Hypotheses

Consider δs as observed values of a set of independent, identically distributed variables
Null hypothesis: the 2 learning systems have the same accuracy
Alternative hypothesis: one system is more accurate than the other
Hypothesis test:

(i) Use paired t-test to determine probability p that mean of δs would arise from null hypothesis

(ii) If p is sufficiently small (usually $p < 0.05$), reject null hypothesis

Comparing systems using a paired t-test:

(i) Calculate sample mean: $\bar{\delta} = (1/n) \sum_{i=1}^{n} \delta_i$

(ii) Calculate t statistic:

$$t = \frac{\bar{\delta}}{\sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (\delta_i - \bar{\delta})^2}}$$

(iii) Determine corresponding p-value: use t in table for t-distribution with $n - l$ degrees of freedom

t-tests compare two learning systems, tests like McNemar's χ^2 test compares two learned models

Scatter plots for pairwise method comparison: can compare thwo methods A and B by plotting ($Aperformance, Bperformanc$
across numerous data sets
Lesion studies: determine relevant contributions to learning system performance by removing (lesioning)
components

5 Computational Learning Theory

5.1 PAC Learning

Computational learning theory concerns itself with:

(i) Sample complexity: how many training examples needed to converge on hypothesis

(ii) Computational complexity: how much computational effort needed for learner to converge

(iii) Mistake bound: how many misclassifications before learner converges on hypothesis

Learning setting composed of:

(i) Set of instances X

(ii) Set of hypotheses H

(iii) Set of possible target concepts C

(iv) Unknown probability distribution D over instances

Learner is given set D of training instances $(x, c(x))$ for some target concept $c \in C$
Learning task is to output hypothesis h modeling c

True error of hypothesis: how often h is wrong on future instances drawn from D:
$error_D(h) = P_{x \in D}(c(x) \neq h(x))$
Traning error: how often h is wrong on instances in training set d
$error_D(h) = P(x \in D)(c(x) \neq h(x)) = (1/|D|) \sum_{x \in D} \delta(c(x) \neq h(x))$

Probably Approximately Correct (PAC) Learning: consider class C of possible target concepts defined over set of instances X of size n, and a learner L using hypothesis space H
C is PAC learnable by L if for all: $c \in C$, distributions D over C, ε and δ such that $0 < \varepsilon, \delta < 0.5$
L will, with probability $> (1 - \delta)$, output hypothesis $h \in H$ such that $error_D(h) \leq \varepsilon$, in time polynomial to $1/\varepsilon, 1/\delta, n, size(c)$
Suppose hypotheses consistent with m training instances
PAC learnability determined by whether

(i) m grows polynomially in the relevant parameters

(ii) Processing time per training exmaple is polynomial

Consistency with respect to training example set $D = \{(x_1, c(x_1), ..., (x_m, c(x_m)))\}$, and hypothesis h

$$consistent(h, D) = \forall((x, c(x)) \in D)h(x) = c(x)$$

Version space: $VS_{H,D} = \{h \in H | consistent(h, D)\}$
Version space $VS_{H,D}$ is ε-exhausted with repsect to concept c and data set D if:
$(\forall h \in VS_{H,D})error_D(h) < \varepsilon$

The probability that VS_{HD} is not ε-exhausted is no greater than $|H|e^{-\varepsilon m}$
Proof: probability that some hypothesis with error greater than ε is consistent with m training instance is $(1-\varepsilon)^m$, there are at most $|H|$ such hypotheses, and since $(1-\varepsilon) \geq e^{-\varepsilon}$ when $0 \leq \varepsilon \leq 1$, the bound is $|H|e^{-\varepsilon m}$

The probability must be reduced below δ: $|H|e^{-\varepsilon m} \geq \delta$
Solving for m yields the number of examples needed for PAC learnability:

$$m \geq \frac{1}{\varepsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

5.2 Hypothesis spaces

Agnostic learning setting: don't assume $c \in H$, learner returns hypothesis h that makes fewest errors in training
For error on training set to be less than $error_D(h) + \varepsilon$, m examples needed, where

$$m \geq \frac{1}{2\varepsilon^2}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

If H is infinite, for measure of hypothesis space complexity, use in place of $|H|$ the largest subset of X for which H can guarantee zero training error regardless of target function

A set of instances D is shattered by a hypothesis space H if and only if for every dichotomy of D, there is a hypothesis in H consistent with the dichotomy
The VC dimension of H defined over instance space X is the size of the largest finite subset of X shattered by H
The VS dimension for finite H: $VC - dim(H) \leq \log_2|H|$
Proof: suppose $VC - dim(H) = d$, and for d instances, there are 2^d different possible labelings. H must represent 2^d hypotheses, so $2^d \leq |H|$, thus $d \leq \log_2|H|$
Using $VC - dim(H)$ as a measure of complexity of H, the bound can be defined as

$$m \geq \frac{1}{\varepsilon}\left(4\log_2\frac{2}{\delta} + 8VC - dim(H)\log_2\frac{13}{\varepsilon}\right)$$

Lower bound on sample complexity: given target concept C

$$m < max\left[\frac{1}{\varepsilon}\log|\frac{1}{\delta}|, \frac{VC - dim(C) - 1}{32\varepsilon}\right]$$

5.3 Mistake Bound

Learning setting (i.e. on-line learning setting): for $t = 1, 2, ...$

(i) Learner receives instance x_t

(ii) Learner predicts $h(x_t)$

(iii) Learner receives label $c(x_t)$ and updates model h

Mistake bound model addresses how many mistakes will be made before the target concept is learned

Mistake bound model analysis of halving algorithm
Halving algorithm is as follows:

(i) $VS_1 = H$

(ii) For $t = 1$ to T:

  (a) Given training instance $(x_t, c(x_t))$, $h'(x_t) = MajorityVote(VS_t, x_t)$

  (b) Eliminate all wrong h from version space (at least half)
  $VS_{t+1} = \{h \in VS_t | h(x_t) = c(x_t)\}$

(iii) Return VS_{t+1}

Maximum number of mistakes is $M_{Halving}(C) = \lfloor log_2|H| \rfloor$

Optimal mistake bounds: suppose C is an arbitrary nonempty concept class
An optimatal mistake bound is defined as $Opt(C) = min_{A \in \text{Learning Algorithms}} M_A(C)$
$Opt(C)$ is the number of mistakes made on the hardest target concept and hardest training sequence by best algorithm
Optimal mistake bound is $VC - dim(C) \leq Opt(C) \leq M_{Halving}(C) \leq log_2(|C|)$

Mistake bound model analysis of weighted majority algorithm
Weighted majority algorithm is as follows: given predictors $A = \{a_1, ..., a_n\}$, learning rate $0 \leq \beta < 1$

(i) For all i, initialize $w_i = 1$

(ii) For each training instance $(x, c(x))$:

  (a) Initialize q_0 and q_0 to 0

  (b) For each predictor a_i: $q_{a_i(x)} = q_{a_i(x)} + w_i$

  (c) If $q_1 > q_0$ then $h(x) = 1$, else if $q_0 > q_1$ then $h(x) = 0$, else $h(x) = random(0, 1)$

  (d) For each predictor a_i: if $a_i(x) \neq c(x)$ then $w_i = \beta w_i$

Weighted majority similar to perceptron (linear separator, robust against noise, mutliplicative weight updates)
If k is minimum number of mistakes made by the best predictor in A on training set D, then maximum number of mistakes for $\beta = 1/2$ is $2.4(k + log_2 n)$

6 Ensemble Methods

7 Neural Networks and Deep Learning

8 Support Vector Machines

9 Reinforcement Learning

10 Rule Learning and Inductive Logic Programming

11 Statistical Relational Learning

12 Bias-Variance Tradeoff

13 Real-Valued Prediciton Methods

14 Dimensionality Reduction