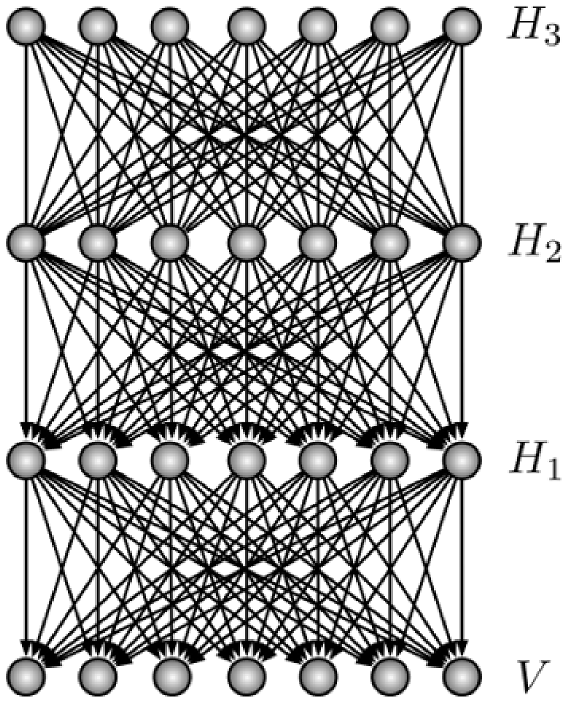
Computer Science 577 Notes
Machine Learning

Mendel C. Mayr

February 4, 2015



Contents

1 Decision Tree Learning

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 $GainRatio = I(D, S)/SplitInfo(D, S)$

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
$I(X, Y) = \sum_{x \in valuesX} \sum_{y \in valuesY} P(x, y) \log_2 P(x, y)/(P(X)P(Y))$

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:

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

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

5 Computational Learning Theory

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