Programming Machine Learning Applications

Lecture Four: Classification and Numeric Prediction, Text Categorization, Bayes, and Decision Trees

Dr. Aleksandar Velkoski

Proximity

k-Nearest-Neighbor Classifier

Interactive Workshop

Review of Lecture Three

Classification

Numeric Prediction

Text Classification

Bayes

Decision Trees

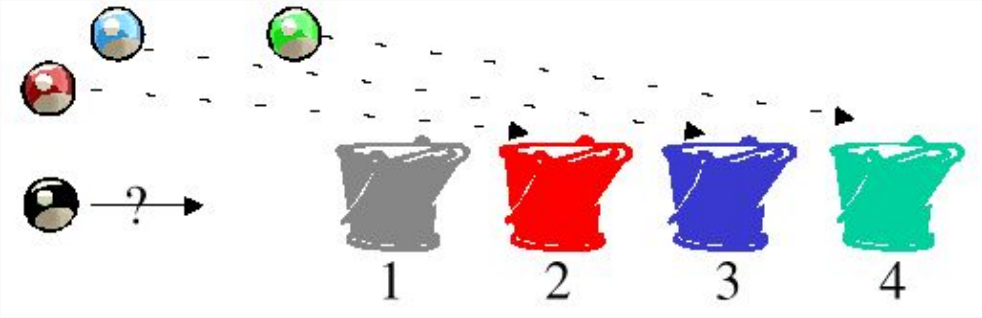
Lecture Four

Classification

What is classification?

The goal of classification is to organize and categorize data in distinct classes

- A model is first created based on the data distribution
- The model is then used to classify new data
- Given the model, a class can be predicted for new data



Classification Task

Given:

- A description of an instance, $x \in X$, where X is the instance language or instance or feature space.
  - Typically, x is a row in a table with the instance/feature space described in terms of features or attributes.
- A fixed set of class or category labels: $C=\{c1, c2,\ldots cn\}$

Determine:

- The class/category of x: $c(x) \in C$, where c(x) is a function whose domain is X and whose range is C.

Learning for Classification

A training example is an instance $x \in X$, paired with its correct class label $c(x)$: $\langle x, c(x) \rangle$ for an unknown classification function, c.

Given a set of training examples, D:

- Find a hypothesized classification function, $h(x)$, such that: $h(x) = c(x)$, for all training instances (i.e., for all $\langle x, c(x) \rangle$ in D). This is called consistency.

Classification Example

Instance language: <size, color, shape>

- size ∈ {small, medium, large}
- color ∈ {red, blue, green}
- shape ∈ {square, circle, triangle}
- C = {positive, negative}

D:

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Hypotheses?

Learning Issues

Many hypotheses can be consistent with the training data

Bias: Criteria other than consistency that is used to select a hypothesis

Classification accuracy (% of instances classified correctly)
- How do we measure accuracy?

Efficiency Issues: ?

Generalization: Hypotheses must correctly classify instances not in training data
- Simply memorizing training data is a consistent hypothesis that does not generalize

Occam's razor: Finding a simple hypothesis helps ensure generalization
- Simplest models tend to be the best models
- The KISS principle

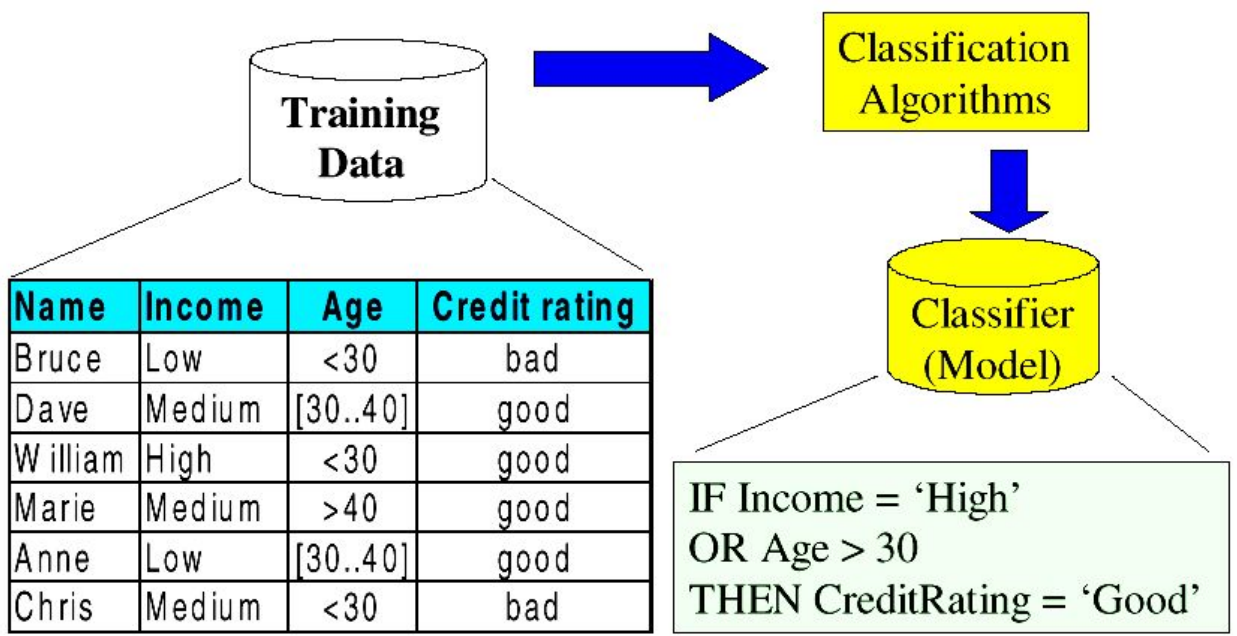
Classification in 3 Steps

1. Model Construction (Learning):
   - Each record (instance, example) is assumed to belong to a predefined class, as determined by one of the attributes
   - This attribute is call the target attribute
   - The values of the target attribute are the class labels
   - The set of all instances used for learning the model is called training set
   - The model may be represented in many forms: decision trees, probabilities, neural networks, ….

2. Model Evaluation (Accuracy):
   - Estimate accuracy rate of the model based on a test set
   - The known labels of test instances are compared with the predicts class from model
   - Test set is independent of training set otherwise over-fitting will occur

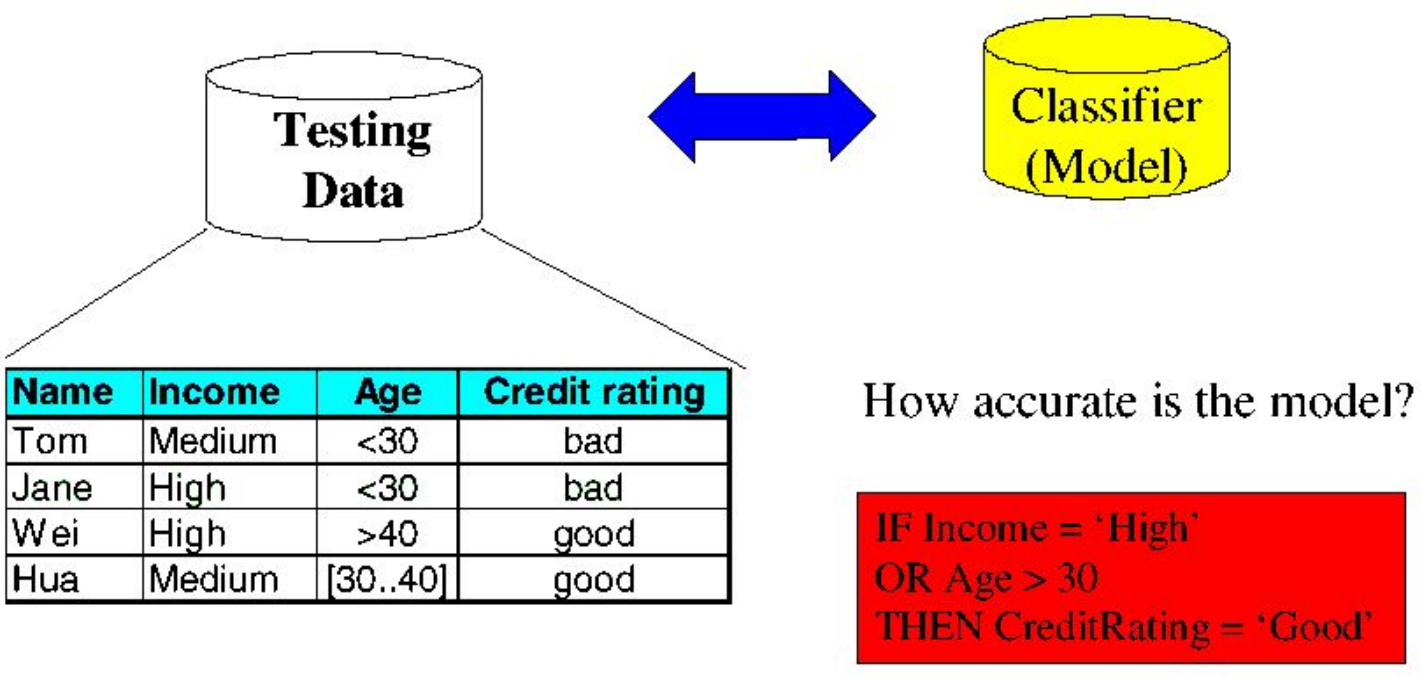
3. Model Use (Classification):
   - The model is used to classify unseen instances (i.e., to predict the class labels for new unclassified instances)
   - Predict the value of an actual attribute

Model Construction



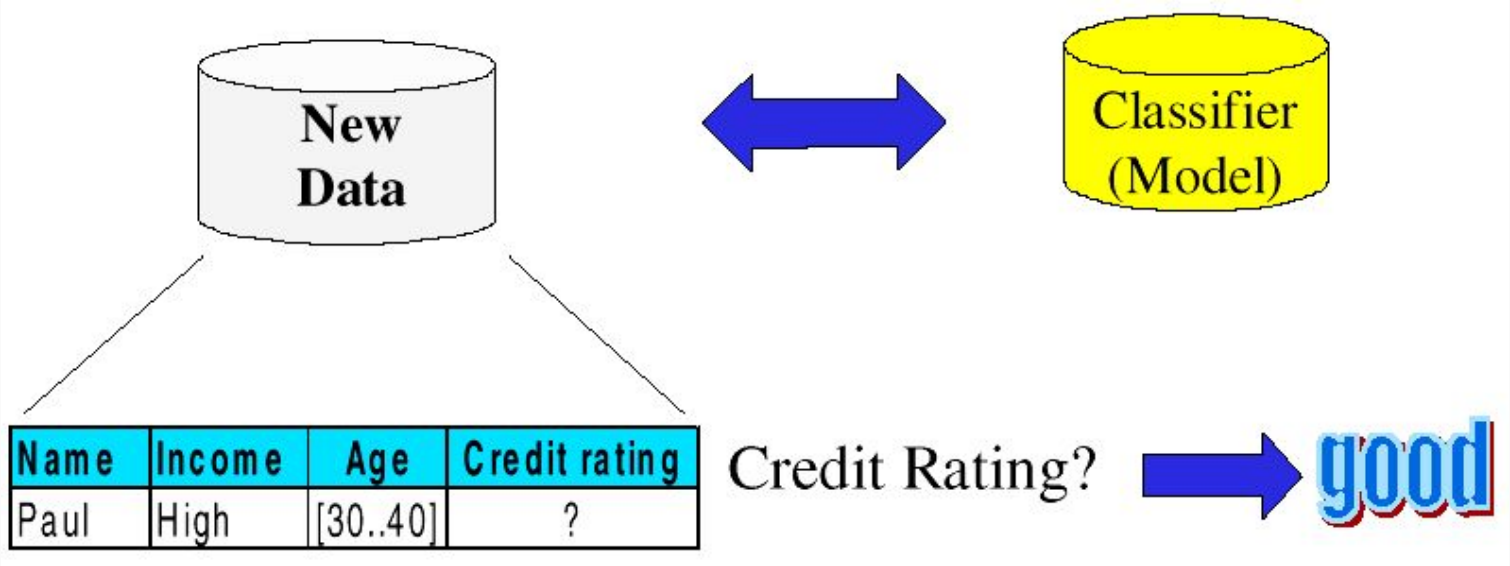
Name	Income	Age	Credit rating
Bruce	Low	<30	bad
Dave	Medium	[30..40]	good
William	High	<30	good
Marie	Medium	>40	good
Anne	Low	[30..40]	good
Chris	Medium	<30	bad

Model Evaluation



Name	Income	Age	Credit rating
Tom	Medium	<30	bad
Jane	High	<30	bad
Wei	High	>40	good
Hua	Medium	[30..40]	good

Model Use


New Data
Classifier (Model)
Name
Income
Age
Credit rating
Paul
High
[30..40]
?
Credit Rating?
good

Classification Methods

Decision Tree Induction

Bayesian Classification

K-Nearest Neighbor

Logistic Regression

Neural Networks

Linear Discriminant Analysis

Support Vector Machines

Genetic Algorithms

Many More ….

Also Ensemble Methods, e.g., Random Forest

Evaluating Models

To train and evaluate models, data are often divided into three sets: the training set, the test set, and the evaluation set

Training Set
- is used to build the initial model
- may need to “enrich the data” to get enough of the special cases

Test Set
- is used to adjust the initial model
- models can be tweaked to be less idiosyncrasies to the training data and can be adapted for a more general model
- idea is to prevent “over-training” (i.e., finding patterns where none exist).

Evaluation Set
- is used to evaluate the model performance

Test and Evaluation Sets

Reading too much into the training set (overfitting)

- common problem with most data mining algorithms
- resulting model works well on the training set but performs poorly on unseen data
- test set can be used to “tweak” the initial model, and to remove unnecessary inputs or features

Evaluation Set is used for final performance evaluation

Insufficient data to divide into three disjoint sets?

- In such cases, validation techniques can play a major role
  - Cross Validation
  - Bootstrap Validation

Cross Validation

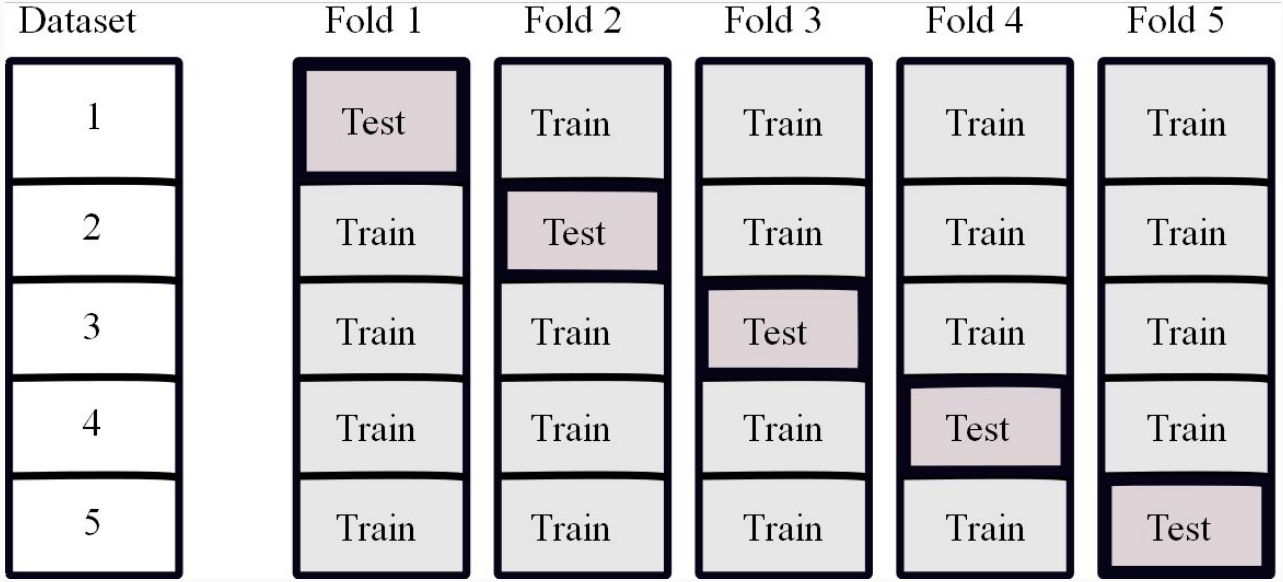
Cross validation is a heuristic that works as follows

- randomly divide the data into n folds, each with approximately the same number of records
- create n models using the same algorithms and training parameters; each model is trained with n-1 folds of the data and tested on the remaining fold
- can be used to find the best algorithm and its optimal training parameter

Steps in Cross Validation

- 1. Divide the available data into a training set and an evaluation set
- 2. Split the training data into n folds
- 3. Select an algorithm and training parameters
- 4. Train and test n models using the n train-test splits
- 5. Repeat step 2 to 4 using different algorithms / parameters and compare model accuracies
- 6. Select the best combination (of algorithms and parameters)
- 7. Use all the training data to train the final model
- 8. Assess the final model using the evaluation set

Example


Dataset
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
1
2
3
4
5
Test
Train
Train
Train
Train
Train
Test
Train
Train
Train
Train
Train
Test
Train
Train
Train
Train
Train
Test
Train
Train
Train
Train
Train
Test

Bootstrap Validation

Based on the statistical procedure of sampling with replacement

- data set of n instances is sampled n times (with replacement) to give another data set of n instances
- since some elements will be repeated, there will be elements in the original data set that are not picked
- these remaining instances are used as the test set

How many instances in the test set?

- Probability of not getting picked in one sampling = 1 - 1/n
- Pr(not getting picked in n samples) = (1 -1/n)n = e-1 = 0.368
- so, for large data set, test set will contain about 36.8% of instances
- to compensate for smaller training sample (63.2%), test set error rate is combined with the re-substitution error in training set:

$$e = (0.632 * e_{\text{test instance}}) + (0.368 * e_{\text{training instance}})$$

Measuring Effectiveness

When the output field is nominal (e.g., in two-class prediction), we use a confusion matrix to evaluate the resulting model

		Predicted Class		
		T	F	Total
Actual Class	T	18	2	20
	F	3	15	18
	Total	21	17	38

- Overall correct classification rate = (18 + 15) / 38 = 87%
- Given T, correct classification rate = 18 / 20 = 90%
- Given F, correct classification rate = 15 / 18 = 83%

Confusion Matrix

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Classifier Accuracy, or recognition rate: percentage of test set instances that are correctly classified

- Accuracy = (TP + TN)/All
- Error rate: 1 – accuracy, or Error rate = (FP + FN)/All

Class Imbalance Problem: One class may be rare, e.g. fraud, or HIV-positive

- Sensitivity: True Positive recognition rate = TP/P
- Specificity: True Negative recognition rate = TN/N

Other Evaluation Metrics

Precision
- % of instances that the classifier predicted as positive that are actually positive

$$precision = \frac{TP}{TP + FP}$$

Recall
- % of positive instances that the classifier predicted correctly as positive
- a.k.a “Completeness”

$$recall = \frac{TP}{TP + FN}$$

Perfect score for both is 1.0, but there is often a trade-off between Precision and Recall

F measure (F1 or F-score)
- harmonic mean of precision and recall

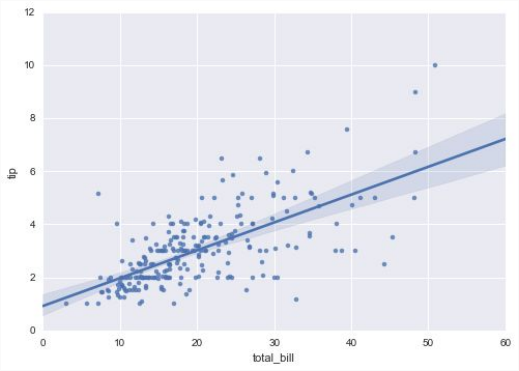
$$F = \frac{2 \times precision \times recall}{precision + recall}$$

Numeric Prediction

What is numeric prediction?

The goal of numeric prediction is to forecast the value of an attribute

- A model is first created based on the data distribution
- The model is then used to predict future or unknown values
- Most common approach: linear regression analysis



What is numeric prediction?

(Numerical) prediction is similar to classification

- construct a model
- use model to predict continuous or ordered value for a given input

Prediction is different from classification

- Classification refers to predict categorical class label
- Prediction models continuous-valued functions

Major method for prediction: regression

- model the relationship between one or more independent or predictor variables and a dependent or response variable

Regression analysis

- Linear and multiple regression
- Non-linear regression
- Other regression methods: generalized linear model, Poisson regression, log-linear models, regression trees

Linear Regression

Linear regression: involves a response variable y and a single predictor variable x

- $y = w_0 + w_1 x$
- w_0 (y-intercept) and w_1 (slope) are regression coefficients

Method of least squares: estimates the best-fitting straight line

$$w_1 = \frac{\sum_{i=1}^{|D|}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|}(x_i - \bar{x})^2} \qquad w_0 = \bar{y} - w_1 \bar{x}$$

Multiple linear regression: involves more than one predictor variable

- Training data is of the form (X1, y1), (X2, y2),..., (X|D|, y|D|)
- Ex. For 2-D data, we may have: y = w0 + w1 x1+ w2 x2
- Solvable by extension of least square method
- Many nonlinear functions can be transformed into the above

Nonlinear Regression

Some nonlinear models can be modeled by a polynomial function

A polynomial regression model can be transformed into linear regression model. For example,

- y = w0 + w1 x + w2 x2 + w3 x3
- is convertible to linear with new variables: x2 = x2, x3= x3
- y = w0 + w1 x + w2 x2 + w3 x3

Other functions, such as power function, can also be transformed to linear model

Some models are intractable nonlinear (e.g., sum of exponential terms)

- possible to obtain least squares estimates through extensive computation on more complex functions

Other Regression Methods

Generalized linear models

- Foundation on which linear regression can be applied to modeling categorical response variables
- Variance of y is a function of the mean value of y, not a constant
- Logistic regression: models the probability of some event occurring as a linear function of a set of predictor variables
- Poisson regression: models the data that exhibit a Poisson distribution

Regression trees and model trees

- Trees to predict continuous values rather than class labels

Other Regression Methods

Regression tree: proposed in CART system (Breiman et al. 1984)

- CART: Classification And Regression Trees
- Each leaf stores a continuous-valued prediction
- It is the average value of the predicted attribute for the training instances that reach the leaf

Model tree: proposed by Quinlan (1992)

- Each leaf holds a regression model—a multivariate linear equation for the predicted attribute
- A more general case than regression tree

Regression and model trees tend to be more accurate than linear regression when instances are not represented well by simple linear models

Other Regression Methods

Prediction Accuracy

- Difference between predicted scores and the actual results (from evaluation set)
- Typically the accuracy of the model is measured in terms of variance (i.e., average of the squared differences)

Common Metrics (pi = predicted target value for test instance i, ai = actual target value for instance i)

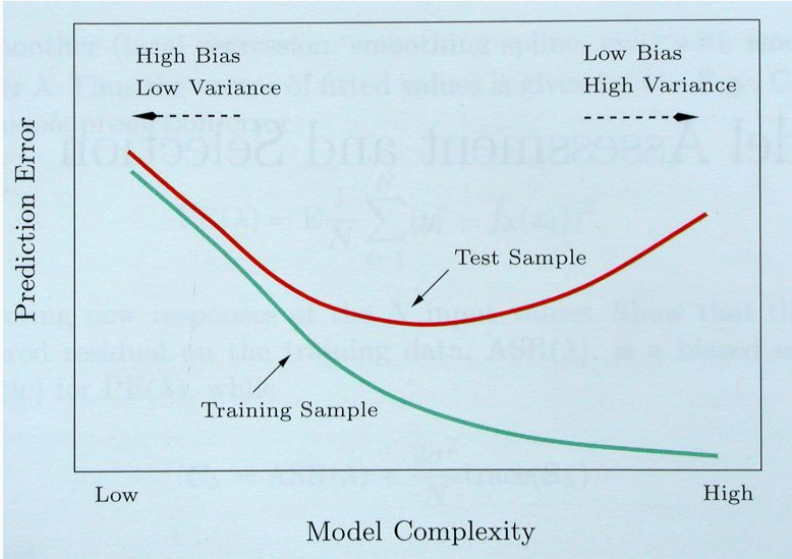
- Mean Absolute Error: Average loss over the test set

$$MAE = \frac{|(p_1 - a_1)| + \ldots + |(p_n - a_n)|}{n}$$

- Root Mean Squared Error: compute the standard deviation (i.e., square root of the covariance between predicted and actual ratings)

$$RMSE = \sqrt{\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}}$$

Bias - Variance Tradeoff Revisited



Bias - Variance Tradeoff Revisited

Possible ways of dealing with high bias

- Get additional features
- More complex model (e.g., adding polynomial terms such as x12, x22 , x1.x2, etc.)
- Use smaller regularization coefficient (in regression models).
- Note: getting more training data won't necessarily help in this case

Possible ways dealing with high variance

- Use more training instances
- Reduce the number of features
- Use simpler models
- Use a larger regularization coefficient.

Text Categorization

About Text Categorization

Assigning documents to a fixed set of categories:

- **Web pages**
  - Recommending pages
  - Yahoo-like classification hierarchies
  - Categorizing bookmarks
- **Newsgroup Messages /News Feeds / Micro-blog Posts**
  - Recommending messages, posts, tweets, etc.
  - Message filtering
- **News articles**
  - Personalized news
- **Email messages**
  - Routing
  - Folderizing
  - Spam filtering

Learning for Text Categorization

Text Categorization is an application of classification

Typical Learning Algorithms:

- Bayesian (naïve)
- Neural network
- Relevance Feedback (Rocchio)
- Nearest Neighbor
- Support Vector Machines (SVM)

Similarity Metrics

Nearest neighbor method depends on a similarity (or distance) metric

Simplest for continuous m-dimensional instance space is Euclidean distance

Simplest for m-dimensional binary instance space is Hamming distance (number of feature values that differ)

For text, cosine similarity of TF-IDF weighted vectors is typically most effective

Basic Automatic Text Processing

Parse documents to recognize structure and meta-data
- e.g. title, date, other fields, html tags, etc.

Scan for word tokens
- lexical analysis to recognize keywords, numbers, special characters, etc.

Stopword removal
- common words such as “the”, “and”, “or” which are not semantically meaningful in a document

Stem words
- morphological processing to group word variants (e.g., “compute”, “computer”, “computing”, “computes”, … can be represented by a single stem “comput” in the index)

Assign weight to words
- using frequency in documents and across documents

Store Index
- Stored in a Term-Document Matrix (“inverted index”) which stores each document as a vector of keyword weights

Basic Automatic Text Processing

tf x idf measure:

- term frequency (tf)
- inverse document frequency (idf) -- a way to deal with the problems of the Zipf distribution
- Recall the Zipf distribution
- Want to weight terms highly if they are
  - frequent in relevant documents … BUT
  - infrequent in the collection as a whole

Goal: assign a tf x idf weight to each term in each document

tf x idf

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

T_k = term k in document D_i

tf_{ik} = frequency of term T_k in document D_i

idf_k = inverse document frequency of term T_k in C

N = total number of documents in the collection C

n_k = the number of documents in C that contain T_k

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

Inverse Document Frequency

IDF provides high values for rare words and low values for common words

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

Example

The initial Term x Doc matrix (Inverted Index)

	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	df	idf = log2(N/df)
T1	0	2	4	0	1	0	3	1.00
T2	1	3	0	0	0	2	3	1.00
T3	0	1	0	2	0	0	2	1.58
T4	3	0	1	5	4	0	4	0.58
T5	0	4	0	0	0	1	2	1.58
T6	2	7	2	1	3	0	5	0.26
T7	1	0	0	5	5	1	4	0.58
T8	0	1	1	0	0	3	3	1.00

Documents represented as vectors of words

tf x idf Term x Doc matrix

	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
T1	0.00	2.00	4.00	0.00	1.00	0.00
T2	1.00	3.00	0.00	0.00	0.00	2.00
T3	0.00	1.58	0.00	3.17	0.00	0.00
T4	1.74	0.00	0.58	2.90	2.32	0.00
T5	0.00	6.34	0.00	0.00	0.00	1.58
T6	0.53	1.84	0.53	0.26	0.79	0.00
T7	0.58	0.00	0.00	2.92	2.92	0.58
T8	0.00	1.00	1.00	0.00	0.00	3.00

Nearest Neighbor Learning

Learning is just storing the representations of the training examples in data set D

Testing instance x:

- Compute similarity between x and all examples in D
- Assign x the category of the most similar examples in D

Does not explicitly compute a generalization or category prototypes (i.e., no “modeling”)

Also called:

- Case-based
- Memory-based
- Lazy learning

Nearest Neighbors for Text

Training:

- For each each training example <x, c(x)> D
  - Compute the corresponding TF-IDF vector, dx, for document x

Test instance y:

- Compute TF-IDF vector d for document y
- For each <x, c(x)> D
  - Let sx = cosSim(d, dx)
- Sort examples, x, in D by decreasing value of sx
- Let N be the first k examples in D. (get most similar neighbors)
- Return the majority class of examples in N

Bayes

Bayesian Learning

Bayes's theorem plays a critical role in probabilistic learning and classification

- Uses prior probability of each class given no information about an item
- Classification produces a posterior probability distribution over the possible classes given a description of an item
- The models are incremental in the sense that each training example can incrementally increase or decrease the probability that a hypothesis is correct. Prior knowledge can be combined with observed data

Given a data instance X with an unknown class label, H is the hypothesis that X belongs to a specific class C

- The conditional probability of hypothesis H given observation X, Pr(H|X), follows the Bayes's theorem:

$$\Pr(H \mid X) = \frac{\Pr(X \mid H)\Pr(H)}{\Pr(X)}$$

Practical difficulty: requires initial knowledge of many probabilities

Basic Concepts in Probability

P(A | B) is the probability of A given B. Assumes that B is all and only information known.

Note that: $P(A \wedge B) = P(A \mid B).P(B)$

Bayes's Rule: Direct corollary of above definition

$$P(A \wedge B) = P(A \mid B).P(B)$$
$$P(B \wedge A) = P(B \mid A).P(A)$$
$$\text{but } P(A \wedge B) = P(B \wedge A)$$
$$\therefore P(A \mid B).P(B) = P(B \mid A).P(A)$$
$$\therefore P(A \mid B) = \frac{P(B \mid A).P(A)}{P(B)}$$

Often written in terms of hypothesis and evidence: $P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$

Basic Concepts in Probability

A and B are independent if:

$$P(A \mid B) = P(A)$$

$$P(B \mid A) = P(B)$$

Therefore, if A and B are independent:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = P(A) \qquad P(A \wedge B) = P(A)P(B)$$

Bayesian Classification

Let set of classes be {c1, c2,…cn}
Let E be description of an instance (e.g., vector representation)
Determine class of E by computing for each class ci

$$P(c_i \mid E) = \frac{P(c_i)P(E \mid c_i)}{P(E)}$$

P(E) can be determined since classes are complete and disjoint:

$$\sum_{i=1}^{n} P(c_i \mid E) = \sum_{i=1}^{n} \frac{P(c_i)P(E \mid c_i)}{P(E)} = 1$$

$$P(E) = \sum_{i=1}^{n} P(c_i)P(E \mid c_i)$$

Bayesian Classification

Need to know:

- Priors: P(ci) and Conditionals: P(E | ci)

P(ci) are easily estimated from data.

- If ni of the examples in D are in ci,then P(ci) = ni / |D|

Assume instance is a conjunction of binary features/attributes:

$$E = e_1 \wedge e_2 \wedge \cdots \wedge e_m$$

Bayesian Classification

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

$E = Outlook = rain \wedge Temp = cool \wedge Humidity = normal \wedge Windy = true$

Naive Bayesian Classification

Problem: Too many possible combinations (exponential in m) to estimate all P(E | ci)

If we assume features/attributes of an instance are independent given the class (ci) (conditionally independent)

$$P(E \mid c_i) = P(e_1 \wedge e_2 \wedge \cdots \wedge e_m \mid c_i) = \prod_{j=1}^{m} P(e_j \mid c_i)$$

Therefore, we then only need to know P(ej | ci) for each feature and category

Estimating Probabilities

Normally, probabilities are estimated based on observed frequencies in the training data.

If D contains ni examples in class ci, and nij of these ni examples contains feature/attribute ej, then:

$$P(e_j \mid c_i) = \frac{n_{ij}}{n_i}$$

If the feature is continuous-valued, P(ej|ci) is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(e_j \mid c_i) = g(e_j, \mu_{c_i}, \sigma_{c_i})$$

Smoothing

Estimating probabilities from small training sets is error-prone:

- If due only to chance, a rare feature, ek, is always false in the training data, ci :P(ek | ci) = 0
- If ek then occurs in a test example, E, the result is that ci: P(E | ci) = 0 and ci: P(ci | E) = 0

To account for estimation from small samples, probability estimates are adjusted or smoothed

Laplace smoothing using an m-estimate assumes that each feature is given a prior probability, p, that is assumed to have been previously observed in a "virtual" sample of size m.

$$P(e_j \mid c_i) = \frac{n_{ij} + mp}{n_i + m}$$

For binary features, p is simply assumed to be 0.5.

Example

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

- Here, we have two classes C1=“yes” (Positive) and C2=“no” (Negative)
- Pr(“yes”) = instances with “yes” / all instances = 9/14
- If a new instance X had outlook=“sunny”, then Pr(outlook=“sunny” | “yes”) = 2/9 (since there are 9 instances with “yes” (or P) of which 2 have outlook=“sunny”)
- Similarly, for humidity=“high”, Pr(humidity=“high” | “no”) = 4/5
- And so on.

Example

Suppose X = < sunny , mild , high , true >

Pr(X | “no”) = 3/5 . 2/5 . 4/5 . 3/5
Pr(X | “yes”) = (2/9 . 4/9 . 3/9 . 3/9)

Outlook	P	N
sunny	2/9	3/5
overcast	4/9	0
rain	3/9	2/5
Tempreature		
hot	2/9	2/5
mild	4/9	2/5
cool	3/9	1/5

Humidity	P	N
high	3/9	4/5
normal	6/9	1/5
Windy		
true	3/9	3/5
false	6/9	2/5

Example

To find out to which class X belongs we need to maximize: Pr(X | Ci).Pr(Ci), for each class Ci (here “yes” and “no”)

$$X = \langle \text{sunny, mild, high, true} \rangle$$

$$\Pr(X \mid \text{“no”}).\Pr(\text{“no”}) = (3/5 \cdot 2/5 \cdot 4/5 \cdot 3/5) \cdot 5/14 = 0.04$$

$$\Pr(X \mid \text{“yes”}).\Pr(\text{“yes”}) = (2/9 \cdot 4/9 \cdot 3/9 \cdot 3/9) \cdot 9/14 = 0.007$$

To convert these to probabilities, we can normalize by dividing each by the sum of the two:

Pr(“no” | X) = 0.04 / (0.04 + 0.007) = 0.85
Pr(“yes” | X) = 0.007 / (0.04 + 0.007) = 0.15

Therefore the new instance X will be classified as “no”.

Spam Example

Training Data

	t1	t2	t3	t4	t5	Spam
D1	1	1	0	1	0	no
D2	0	1	1	0	0	no
D3	1	0	1	0	1	yes
D4	1	1	1	1	0	yes
D5	0	1	0	1	0	yes
D6	0	0	0	1	1	no
D7	0	1	0	0	0	yes
D8	1	1	0	1	0	yes
D9	0	0	1	1	1	no
D10	1	0	1	0	1	yes

Term	P(t|no)	P(t|yes)
t1	1/4	4/6
t2	2/4	4/6
t3	2/4	3/6
t4	3/4	3/6
t5	2/4	2/6

P(no) = 0.4
P(yes) = 0.6

New email *x* containing t1, t4, t5 ≡ $x = \langle 1, 0, 0, 1, 1 \rangle$

Should it be classified as spam = "yes" or spam = "no"?
Need to find P(yes | *x*) and P(no | *x*) …

Additional Example

New email *x* containing t1, t4, t5

$x = \langle 1, 0, 0, 1, 1 \rangle$

Term	P(t|no)	P(t|yes)
t1	1/4	4/6
t2	2/4	4/6
t3	2/4	3/6
t4	3/4	3/6
t5	2/4	2/6

P(no) = 0.4
P(yes) = 0.6

$P(yes \mid x) = [4/6 * (1-4/6) * (1-3/6) * 3/6 * 2/6] * P(yes) / P(x)$
$= [0.67 * 0.33 * 0.5 * 0.5 * 0.33] * 0.6 / P(x) = 0.11 / P(x)$

$P(no \mid x) = [1/4 * (1-2/4) * (1-2/4) * 3/4 * 2/4] * P(no) / P(x)$
$= [0.25 * 0.5 * 0.5 * 0.75 * 0.5] * 0.4 / P(x) = 0.019 / P(x)$

To get actual probabilities need to normalize: note that $P(yes \mid x) + P(no \mid x)$ must be 1

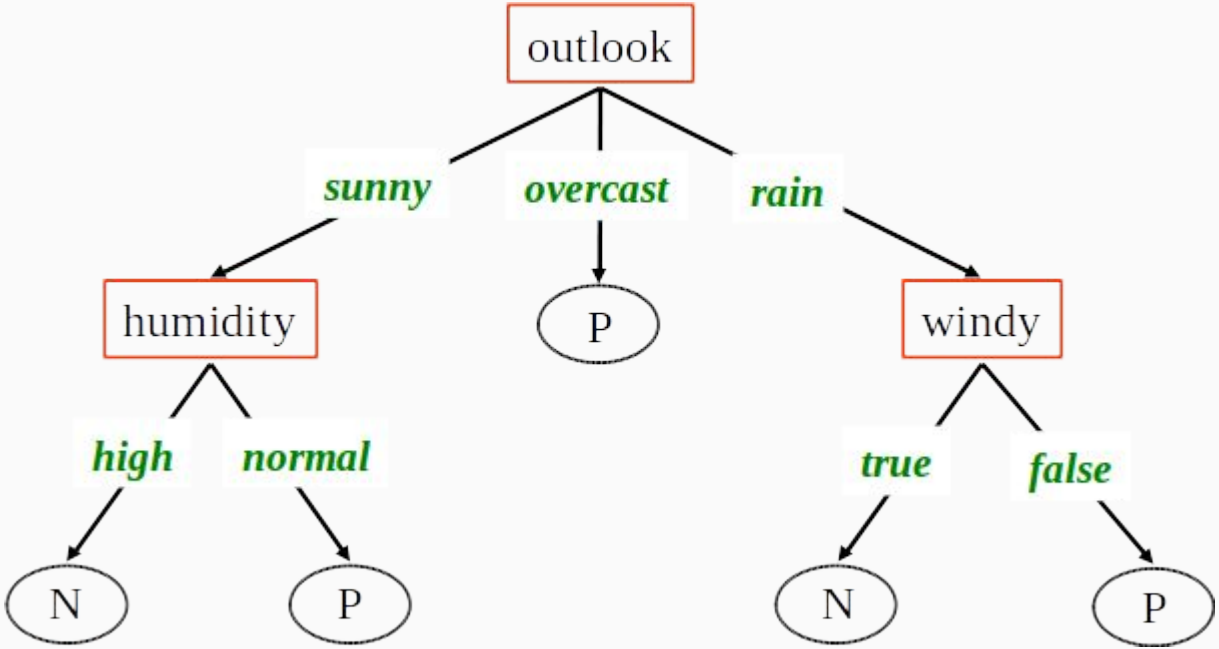
$0.11 / P(x) + 0.019 / P(x) = 1 \quad \Rightarrow \quad P(x) = 0.11 + 0.019 = 0.129$

So: $P(yes \mid x) = 0.11 / 0.129 = 0.853$

$P(no \mid x) = 0.019 / 0.129 = 0.147$

Decision Trees

About Decision Trees



Instance Language Classification

Example: "is it a good day to play golf?"

4 a set of attributes and their possible values:

outlook	sunny, overcast, rain
temperature	cool, mild, hot
humidity	high, normal
windy	true, false

A particular *instance* in the training set might be:

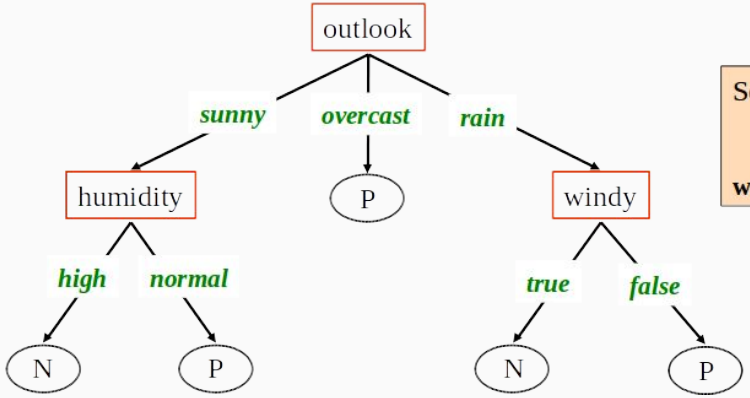
<**overcast, hot, normal, false**>: **play**

In this case, the target class is a binary attribute, so each instance represents a positive or a negative example.

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Using Decision Trees for Classification

- **Examples can be classified as follows**
  - 1. look at the example's value for the feature specified
  - 2. move along the edge labeled with this value
  - 3. if you reach a leaf, return the label of the leaf
  - 4. otherwise, repeat from step 1
- **Example (a decision tree to decide whether to go on a picnic):**



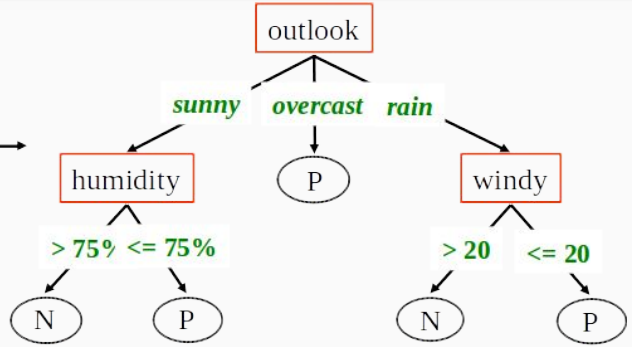
So a new instance:

<rainy, hot, normal, true>: ?

will be classified as "noplay"

Decision Trees and Decision Rules

If attributes are continuous, internal nodes may test against a threshold.



Each path in the tree represents a decision rule:

Rule1:
If (outlook="sunny") AND (humidity<=0.75)
Then (play="yes")

Rule2:
If (outlook="rainy") AND (wind>20)
Then (play="no")

Rule3:
If (outlook="overcast")
Then (play="yes")

• • •

Top Down Tree Generation

The basic approach usually consists of two phases

- Tree construction
  - At the start, all the training instances are at the root
  - Partition instances recursively based on selected attributes
- Tree pruning (to improve accuracy)
  - Remove tree branches that may reflect noise in the training data and lead to errors when classifying test data

Basic Steps in Decision Tree Construction

- Tree starts a single node representing all data
- If instances are all same class then node becomes a leaf labeled with class label
- Otherwise, select feature that best distinguishes among instances
  - Partition the data based the values of the selected feature (with each branch representing one partitions)
- Recursion stops when:
  - instances in node belong to the same class (or if too few instances remain)
  - There are no remaining attributes on which to split

Top Down Tree Generation

Decision Tree Learning Method (ID3)

- Input: a set of training instances S, a set of features F
- 1. If every element of S has a class value “yes”, return “yes”; if every element of S has class value “no”, return “no”
- 2. Otherwise, choose the best feature f from F (if there are no features remaining, then return failure);
- 3. Extend tree from f by adding a new branch for each attribute value of f
  - 3.1. Set F’ = F – {f},
- 4. Distribute training instances to leaf nodes (so each leaf node n represents the subset of examples Sn of S with the corresponding attribute value
- 5. Repeat steps 1-5 for each leaf node n with Sn as the new set of training instances and F’ as the new set of attributes

Main Question:

- how do we choose the best feature at each step?

Choosing the Best Feature

Use Information Gain to find the "best" (most discriminating) feature
Assume there are two classes, P and N (e.g, P = "yes" and N = "no")

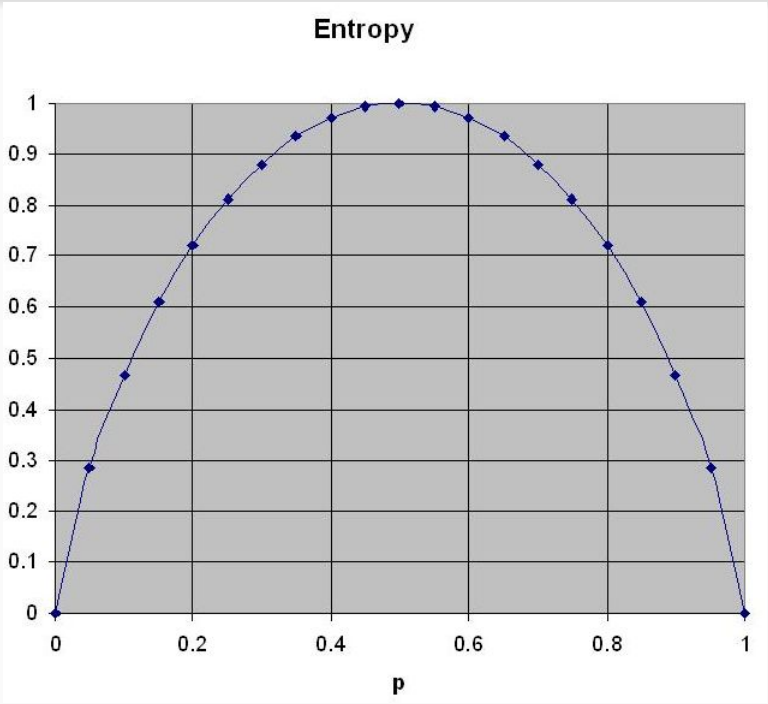
- Let the set of instances S (training data) contains p elements of class P and n elements of class N
- The amount of information, needed to decide if an arbitrary example in S belongs to P or N is defined in terms of entropy, I(p,n):

$$I(p,n) = -\Pr(P)\log_2 \Pr(P) - \Pr(N)\log_2 \Pr(N)$$

- Note that Pr(P) = p / (p+n) and Pr(N) = n / (p+n)

- In other words, entropy of a set on instances S is a function of the probability distribution of classes among the instances in S.

Entropy



Multi-Class Problems

- More generally, if we have m classes, c1, c2, …, cm , with s1, s2, …, sm as the numbers of instances from S in each class, then the entropy is:

$$I$$

$$I(s_1, s_2, \ldots, s_n) = -\sum_{i=1}^{m} p_i \log_2 p_n$$

- where pi is the probability that an arbitrary instance belongs to the class ci.

Information Gain

- Now, assume that using attribute A a set S of instances will be partitioned into sets S1, S2 , …, Sv each corresponding to distinct values of attribute A.
- If Si contains pi cases of P and ni cases of N, the entropy, or the expected information needed to classify objects in all subtrees Si is

$$E(A) = \sum_{i=1}^{v} \Pr(S_i) I(p_i, n_i) \quad \text{where,} \quad \Pr(S_i) = \frac{|S_i|}{|S|} = \frac{p_i + n_i}{p + n}$$

The probability that an arbitrary instance in S belongs to the partition S_i

- The encoding information that would be gained by branching on A:

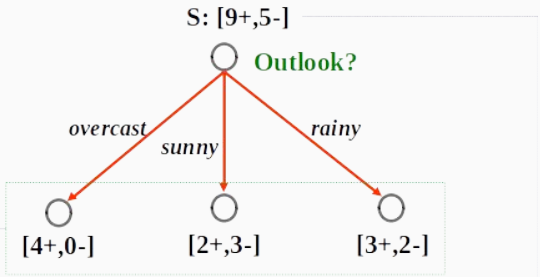
$$Gain(A) = I(p, n) - E(A)$$

- At any point we want to branch using an attribute that provides the highest information gain.

Attribute Selection Example

- The "Golf" example: what attribute should we choose as the root?

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	*No*
D2	sunny	hot	high	strong	*No*
D3	overcast	hot	high	weak	*Yes*
D4	rain	mild	high	weak	*Yes*
D5	rain	cool	normal	weak	*Yes*
D6	rain	cool	normal	strong	*No*
D7	overcast	cool	normal	strong	*Yes*
D8	sunny	mild	high	weak	*No*
D9	sunny	cool	normal	weak	*Yes*
D10	rain	mild	normal	weak	*Yes*
D11	sunny	mild	normal	strong	*Yes*
D12	overcast	mild	high	strong	*Yes*
D13	overcast	hot	normal	weak	*Yes*
D14	rain	mild	high	strong	*No*



I(9,5) = -(9/14).log(9/14) - (5/14).log(5/14)
= 0.94

I(4,0) = -(4/4).log(4/4) - (0/4).log(0/4)
= 0

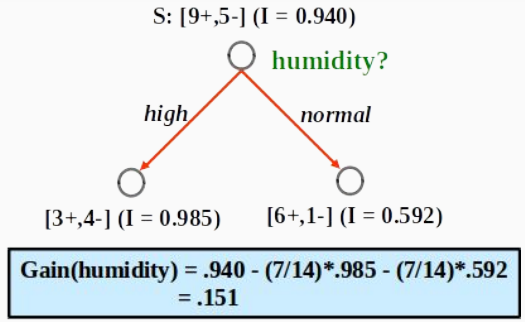
I(2,3) = -(2/5).log(2/5) - (3/5).log(3/5)
= 0.97

I(3,2) = -(3/5).log(3/5) - (2/5).log(2/5)
= 0.97

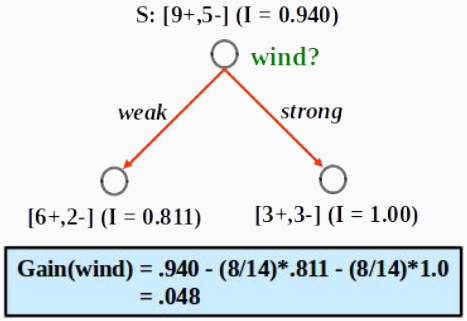
Gain(outlook) = .94 - (4/14)*0
- (5/14)*.97
- (5/14)*.97
= .24

Attribute Selection Example

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	*No*
D2	sunny	hot	high	strong	*No*
D3	overcast	hot	high	weak	*Yes*
D4	rain	mild	high	weak	*Yes*
D5	rain	cool	normal	weak	*Yes*
D6	rain	cool	normal	strong	*No*
D7	overcast	cool	normal	strong	*Yes*
D8	sunny	mild	high	weak	*No*
D9	sunny	cool	normal	weak	*Yes*
D10	rain	mild	normal	weak	*Yes*
D11	sunny	mild	normal	strong	*Yes*
D12	overcast	mild	high	strong	*Yes*
D13	overcast	hot	normal	weak	*Yes*
D14	rain	mild	high	strong	*No*



Gain(humidity) = .940 - (7/14)*.985 - (7/14)*.592 = .151

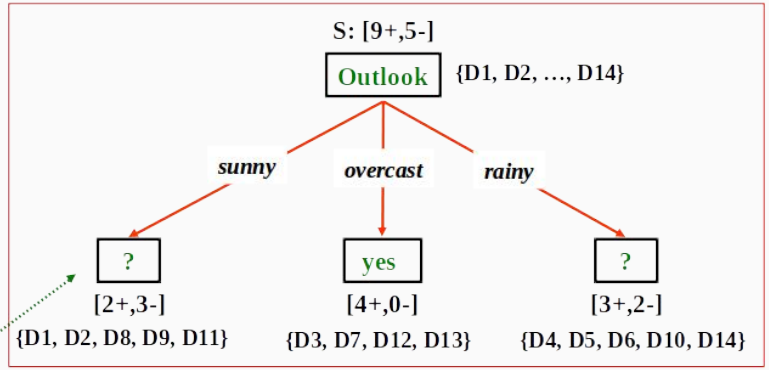


Gain(wind) = .940 - (8/14)*.811 - (8/14)*1.0 = .048

So, classifying examples by humidity provides more information gain than by wind. Similarly, we must find the information gain for "temp". In this case, however, you can verify that outlook has largest information gain, so it'll be selected as root

Attribute Selection Example

- **Partially learned decision tree**



- **which attribute should be tested here?**

S_{sunny} = {D1, D2, D8, D9, D11}

Gain(S_{sunny}, humidity) = .970 - (3/5)*0.0 - (2/5)*0.0 = .970

Gain(S_{sunny}, temp) = .970 - (2/5)*0.0 - (2/5)*1.0 - (1/5)*0.0 = .570

Gain(S_{sunny}, wind) = .970 - (2/5)*1.0 - (3/5)*.918 = .019

Other Attribute Selection Measures

Gain ratio:

- Information Gain measure tends to be biased in favor attributes with a large number of values
- Gain Ratio normalizes the Information Gain with respect to the total entropy of all splits based on values of an attribute
- Used by C4.5 (the successor of ID3)
- But, tends to prefer unbalanced splits (one partition much smaller than others)

Gini index:

- A measure of impurity (based on relative frequencies of classes in a set of instances)
  - The attribute that provides the smallest Gini index (or the largest reduction in impurity due to the split) is chosen to split the node
- Possible Problems:
  - Biased towards multivalued attributes; similar to Info. Gain.
  - Has difficulty when # of classes is large

Overfitting and Tree Pruning

Overfitting: An induced tree may overfit the training data

- Too many branches, some may reflect anomalies due to noise or outliers
- Some splits or leaf nodes may be the result of decision based on very few instances, resulting in poor accuracy for unseen instances

Two approaches to avoid overfitting

- Prepruning: Halt tree construction early-do not split a node if this would result in the error rate going above a pre-specified threshold
  - Difficult to choose an appropriate threshold
- Postpruning: Remove branches from a “fully grown” tree
  - Get a sequence of progressively pruned trees
  - Use a test data different from the training data to measure error rates
  - Select the “best pruned tree”

Enhancements

Allow for continuous-valued attributes

- Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals

Handle missing attribute values

- Assign the most common value of the attribute
- Assign probability to each of the possible values

Attribute construction

- Create new attributes based on existing ones that are sparsely represented
- This reduces fragmentation, repetition, and replication

Wrapping-up the Lecture

Questions

What is the difference between classification and numeric prediction?

What is the difference between Test Set and Validation Set?

What metrics can you use to evaluate model performance?

What is the intuition of tf idf?