Background: Computational Biology

Algorithmic models provide for biological sciences what calculus provided for classical physics.

The language of computation is the best language we have so far for describing how information is encoded, stored, manipulated and used by biological systems.

Central problem: Given genomic sequences – text in a language with known alphabet but unknown syntax and semantics, and some additional clues, discover the syntax and semantics!

Goal is to develop information processing or computational models of biological processes (protein folding, gene regulation, protein-protein interaction).

Background: Bioinformatics

Transformation of Biology from a data poor science to a data rich science:
- High throughput technologies for data acquisition
- Digital technologies for data storage
- Communication technologies for data transfer
- Computing technologies for data processing

In principle, it is possible to gather, store, access, and analyze large volumes of data (e.g., sequence data, structure data, expression data)

The focus of bioinformatics is on the design and implementation of software tools for data-driven knowledge discovery in data rich biological sciences.
Computer-Assisted Knowledge Discovery in Bioinformatics

Challenges in transforming data into knowledge
- Autonomous, heterogeneous, distributed data sources
- Massive quantity of data in some areas and scarcity of data in others
- Noisy measurements, inconsistencies
- Complex a priori unknown relationships

Machine learning algorithms offer some of the most flexible, powerful, and cost-effective approaches to discovery of complex a priori unknown relationships from large data sets

Intelligent Data Understanding System (INDUS)

Representative Machine Learning Applications in Bioinformatics and Computational Biology
- Gene finding
- Ribosome binding site identification
- Promoter identification
- Prediction of protein structural features
- Protein binding site identification
- Prediction of protein function
- Genetic network inference
- Cancer diagnosis
- Gene annotation
Synthesis of protein function classifiers

- Data set of proteins with known function
- Representation of labeled sequences
- Training Set
- Learning algorithm
- Classifier
- Test set
- Representation of a Novel Protein sequence

Machine Learning

- Background and Motivation
- What is learning?
- What is machine learning?
- How can we specify a learning problem?
- Examples of learning algorithms
- Representative applications in bioinformatics and computational biology

Background: Artificial Intelligence

Computation : Cognition :: Calculus: Physics

Algorithms or computation or information processing provide for study of cognition what calculus provided for physics

We have a theory of intelligent behavior when we have precise information processing models (computer programs) that produce such behavior
Motivation

Practical
- Intelligent behavior requires knowledge
- Explicitly specifying the knowledge needed for specific tasks is hard, and often infeasible
- If we can program computers to learn from experience, we can
  - Dramatically enhance the usability of software e.g., personalized information assistants
  - Dramatically reduce the cost of software development e.g., for medical diagnosis
  - Automate data driven discovery

Motivation

Scientific
Information processing models can provide useful insights into
- How humans and animals learn
- Information requirements of learning tasks
- The precise conditions under which certain learning goals are achievable
- Inherent difficulty of learning tasks
- How to improve learning – e.g., value of active versus passive learning
- Computational architectures for learning

Machine Learning is an interdisciplinary field

- Scientific Discovery
- Bioinformatics
- Security Informatics
- Data Mining
- Government
- Medical Informatics
- Commerce
- Smart Artifacts
- Machine Learning
- Computer Science
- Cognitive Science
- Statistics
- Mathematics
Machine Learning: Application Areas

Bioinformatics and Computational Biology
Human Computer Interaction and Pervasive Computing
Economics and Commerce
Computer Assisted Collaborative Learning and Discovery
Intelligent Information Infrastructure
Digital Government
Cognitive Modeling
Robotics
Engineering
Security Informatics

Machine Learning: Contributing Disciplines

Computer Science — Artificial Intelligence, Algorithms and Complexity, Databases, Data Mining
Statistics — Statistical Inference, Experiment Design, Exploratory Data Analysis
Mathematics — Abstract Algebra, Logic, Information Theory, Probability Theory
Psychology and Neuroscience — Behavior, Perception, Learning, Memory, Problem solving
Philosophy — Ontology, Epistemology, Philosophy of Mind, Philosophy of Science

What is learning?

Learning is a process by which the learner improves its performance on a task or a set of tasks as a result of experience within some environment.
Learning = Inference + Memorization (in some context)

Inference
Inductive $p(a), p(b), p(c)\ldots \exists x p(x)$
Deductive
Abductive
What is Machine Learning?

A program $M$ is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$ if its performance as measured by $P$ on tasks in $T$ in an environment $Z$ improves with experience $E$.

Example 1
- $T$ – cancer diagnosis
- $E$ – a set of diagnosed cases
- $P$ – accuracy of diagnosis on new cases
- $Z$ – noisy measurements, occasionally misdiagnosed training cases
- $M$ – a program that runs on a general purpose computer

Example 2
- $T$ – solving calculus problems
- $E$ – practice problems + rules of calculus
- $P$ – score on a test

Example 3
- $T$ – driving on the interstate
- $E$ – a sequence of sensor measurements and driving actions recorded while observing an expert driver
- $P$ – mean distance traveled before an error as judged by a human expert

A general framework for learning
Learning

Data

Learning = Inference + Memorization

Knowledge

Types of learning

Rote Learning – useful when it is less expensive to store and retrieve some information than to compute it

Learning from Instruction – transform instructions into operationally useful knowledge

Learning from Examples (and counter-examples) – extract predictive or descriptive regularities from data

Learning from Deduction (and explanation) – generalize instances of deductive problem-solving

Learning from Exploration – learn to choose actions that maximize reward

Why should Machines Learn?

Some tasks are best specified by example (e.g., credit risk assessment, face recognition)

Some tasks are best shown by demonstration (e.g., landing an airplane)

Buried in large volume of data are useful predictive relationships (data mining)

The operating environment of certain types of software (user characteristics, distribution of problem instances) may not be completely known at design time

Environment changes over time – ability of software to adapt to changes would enhance usability
Examples of Applications of Machine Learning

Data Mining –
  - Using historical data to improve decisions
    - credit risk assessment, diagnosis, electric power usage prediction
  - Using scientific data to acquire knowledge
    - in computational molecular biology
  - Software applications that are hard to program
    - autonomous driving, face recognition, speech recognition
  - Self-customizing programs
    - newsreader that learns user interests

Designing a learning program for a task

Experience – What experiences are available?
  - Data – in medical diagnosis, expert diagnosed cases, feedback
  - How representative is the experience?

Critic – can the learner ask questions?
  - What type of questions?
    - How am I doing? – performance query
    - How would you diagnose X? – example based query
    - Why was I wrong? – explanation

Designing a learning program

Performance element –
  - How is the learned knowledge encoded?
    - rules, probabilities, programs
  - How is the learned knowledge used?
    - e.g. matching rules
  - What is the performance measure?
  - How is performance measured?
    - online? batch?
Designing a learning program

Learning element
What is the learning algorithm?
– search for a set of classification rules that are likely to perform well on novel cases (how?)
– estimate a class conditional probability distribution (how?)

Environment
Deterministic or stochastic?
Noisy or noise free?

Machine Learning
Learning involves synthesis or adaptation of computational structures
Functions
Logic programs
Rules
Grammars
Probability distributions
Action policies
Behaviors

Machine Learning = (Statistical) Inference + Data Structures + Algorithms

Learning input – output functions
Target function \( f \) – unknown to the learner – \( f \in F \)
Learner’s hypothesis about what \( f \) might be – \( h \in H \)

\( H \) – hypothesis space
Instance space – \( X \) – domain of \( f, h \)
Output space – \( Y \) – range of \( f, h \)

Example – an ordered pair \( (x, y) \) where \( x \in X \) and \( f(x) = y \in Y \)

\( F \) and \( H \) may or may not be the same!

Training set \( E \) – a multi set of examples
Learning algorithm \( L \) – a procedure which given some \( E \), outputs an \( h \in H \)
Learning input – output functions

- Must choose
  - Hypothesis language
  - Instance language
  - Semantics associated with both
- Machines can learn only functions that have finite descriptions or representations if we require learning programs to be halting programs
- Examples: "Tom likes science fiction horror films"  
  "E = ma"

Inductive Learning

**Premise** – A hypothesis (e.g., a classifier) that is consistent with a sufficiently large number of representative training examples is likely to accurately classify novel instances drawn from the same universe

- We can prove that this is an optimal approach (under appropriate assumptions)
- With stronger bias, there is less reliance on the training data

Learning and Bias

Example

There is an infinite number of functions that match any finite number of training examples!

Bias free function learning is impossible!
Learning and Bias

Suppose $H = \text{set of all } n$-input Boolean functions

$$|H| = 2^{2^n}$$

Suppose the learner is unbiased

$H_v = \text{version space} \rightarrow \text{the subset of } H \text{ not yet ruled out by the learner}$

- Biased learner
- Bias free learner

Number of unique examples already seen

Learning and Bias

Weaker bias $\Rightarrow$ more open to experience, flexible
- more expressive hypothesis representation

Occam's razor
- simpler hypotheses preferred
- Linear fit preferred to quadratic fit assuming both yield relatively good fit over the training examples

Learning in practice requires a tradeoff between complexity of hypothesis and goodness of fit
The world according to Agent Bob

An atomic event or world state is a complete specification of the state of the agent's world.

Event set is a set of mutually exclusive and exhaustive possible world states (relative to an agent's representational commitments and sensing abilities)

From the point of view of an agent Bob who can sense only 3 colors and 2 shapes, the world can be in only one of 6 states.

Atomic events (world states) are:
- mutually exclusive
- exhaustive
Probability as a subjective measure of belief

Suppose there are 3 agents – Adrian, Anna, Jun, in a world where a dice has been tossed. Adrian observes that the outcome is a “6” and whispers to Anna that the outcome is “even” but Jun knows nothing about the outcome.

Set of possible mutually exclusive and exhaustive world states = {1, 2, 3, 4, 5, 6}
Set of possible states of the world based on what Anna knows = {2, 4, 6}

Probability is a measure over all of the world states that are possible, or simply, possible worlds, given what the agent knows.

\( \text{Possibleworlds}_{\text{Adrian}} = \{6\}, \text{Possibleworlds}_{\text{Anna}} = \{2, 4, 6\} \)
\( \text{Possibleworlds}_{\text{Jun}} = \{1, 2, 3, 4, 5, 6\} \)
\( \text{Pr}_{\text{Adrian}}(\text{worldstate} = 6) = 1 \)
\( \text{Pr}_{\text{Anna}}(6) = \frac{1}{3} \)
\( \text{Pr}_{\text{Jun}}(6) = \frac{1}{6} \)

Random variables

The "domain" of a random variable is the set of values it can take. The values are mutually exclusive and exhaustive.

The domain of a Boolean random variable X is \{true, false\} or \{1, 0\}

Discrete random variables take values from a countable domain.

The domain of the random variable Color may be \{Red, Green\}.

If \( E = \{(\text{Red, Square}), (\text{Green, Circle}), (\text{Red, Circle}), (\text{Green, Square})\} \), the proposition (Color = Red) is True in the world states \{(Red, Square), (Red, Circle)\}.
Defining Probability – Probability Spaces

Definition: Finite Probability Space \( (\mathcal{E}, P) \)

Let \( \mathcal{E} \) be a finite set and let \( P : \mathcal{E} \rightarrow \mathbb{R}^+ \) be a function from \( \mathcal{E} \) to non-negative real numbers such that \( \sum_{\mathcal{E}} P(e) = 1 \).

Then we refer to \( \mathcal{E} \) as an event set and \( P(e) \) as the probability that event \( e \) occurs, or simply, the probability of \( e \).

The elements of \( \mathcal{E} \) are called simple events or elementary events and \( P \) is a probability distribution.

Example: \( \mathcal{E} = \{H, T\} \); \( P(H) = P(T) = \frac{1}{2} \)

Probability of compound events

Let \( (\mathcal{E}, P) \) be a finite probability space.

A compound event \( A \) corresponds to a subset of \( \mathcal{E} \) (a possible world).

\[ P(A) = \sum_{e \in A} P(e) \]

We say that \( A \) occurs if some \( e \in A \) occurs.

Note: Note the "overloading" of the function \( P \).

Fundamental theorem of probability

Let \( (\mathcal{E}, P) \) be a finite probability space. Then

a. if \( A \subseteq B \subseteq \mathcal{E} \), \( 0 \leq P(A) \leq P(B) \leq 1 \)

b. if \( A, B \subseteq \mathcal{E} \), \( P(A \cap B) = P(A) + P(B) - P(A \cup B) \)

c. if \( A, B \subseteq \mathcal{E} \), \( P(A \cup B) = P(A) + P(B) - P(A \cap B) \)

d. if \( A_i \subseteq \mathcal{E} \); \( 1 \leq i \leq n \); and \( \forall i \neq j \), \( A_i \cap A_j = \emptyset \),

\[ P \left( \bigcup_{i=1}^{n} A_i \right) = \sum_{i=1}^{n} P(A_i) \]
Example: Suppose I have two coins – one a normal fair coin, and the other with 2 heads. I pick a coin at random and tell you that the side I am looking at is a head. What is the probability that I am looking at a normal coin?

Label the sides h, h' so that the side labeled h corresponds to a head on the normal coin and both sides correspond to a head on the 2-sided coin

n, t - normal versus 2-sided

\( i = n, t \rightarrow (h, h') \)

Compound events N, H

\( N = \{ (n, h), (n, h') \} \) (selecting the normal coin)

\( H = \{ (n, h), (t, h), (t, h') \} \) (selecting a head)

\[ P(N|H) = \frac{P(N \cap H)}{P(H)} = \frac{1/4}{3/4} = \frac{1}{3} \]

Conditional probability and Bayes Rule

\[ P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \]

\[ P(A_1 \cap \ldots \cap A_n) = P(A_1)P(A_2|A_1)P(A_3|A_1,A_2)\ldots P(A_n|A_1,A_2,\ldots,A_{n-1}) \]

In the case of random variables X, Y

\[ P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X) \]

denotes a set of equations corresponding to possible assignments of values to the random variables
Bayes Theorem

Does patient have cancer or not?

A patient takes a lab test and the result comes back positive. The test returns a correct positive result in only 98% of the cases in which the disease is actually present, and a correct negative result in only 97% of the cases in which the disease is not present. Furthermore, .008 of the entire population have this cancer.

\[
P(\text{cancer}) = \quad P(\neg \text{cancer}) = \]
\[
P(+) | \text{cancer} = \quad P(+) | \neg \text{cancer} =
\[
P(+) | \neg \text{cancer} = \quad P(+) | \neg \text{cancer} =
\]

\[
P(\text{cancer}) = P(\neg \text{cancer}) =
\]
\[
P(+ \cap \text{cancer}) = P(+) | \text{cancer} \cdot P(\text{cancer})
\]
\[
P(+ \cap \neg \text{cancer}) = P(+) | \neg \text{cancer} \cdot P(\neg \text{cancer})
\]

\[
P(\text{cancer} | +) = \frac{P(+ \cap \text{cancer})}{P(+)} = \frac{P(+) | \text{cancer} \cdot P(\text{cancer})}{P(+)}
\]
\[
P(\neg \text{cancer} | +) = \frac{P(+ \cap \neg \text{cancer})}{P(+)} = \frac{P(+) | \neg \text{cancer} \cdot P(\neg \text{cancer})}{P(+)}
\]

The patient, more likely than not, does not have cancer.
Random Variables

A random variable defines a set of compound events which form a partition of the event set.

\[ E = \{(\text{red, square}), (\text{green, square}), (\text{red, circle}), (\text{green, circle})\} \]

The random variable Color with domain \( S = \{\text{red, green}\} \) partitions \( E \) into \( E_{\text{red}} \) and \( E_{\text{green}} \).

\[ E_{\text{red}} = \{(\text{red, square}), (\text{red, circle})\} \]
\[ E_{\text{green}} = \{(\text{green, square}), (\text{green, circle})\} \]

\[ E = E_{\text{red}} \cup E_{\text{green}} \quad \text{and} \quad E_{\text{red}} \cap E_{\text{green}} = \emptyset \]

Probability Distribution of Random Variables

If \( X \) is a random variable with a finite domain, we use \( P(X) \) to denote the unconditional probabilities associated with each possible value of \( X \).

Example: \( \text{Domain(Height)} = \{\text{tall, medium, short}\} \)
\( \text{Domain(Play)} = \{\text{yes, no}\} \)

Joint Distribution \( P(\text{Height}, \text{Play}) \) is a 3×2 table of entries that sum to 1.

Inference using the joint distribution

<table>
<thead>
<tr>
<th></th>
<th>ache</th>
<th>¬ache</th>
</tr>
</thead>
<tbody>
<tr>
<td>cavity</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>¬cavity</td>
<td>0.1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

\[ P(\text{cavity}) = P(\text{cavity, ache}) + P(\text{cavity, ¬ache}) \]
**Effect of Evidence on Possible worlds**

Evidence \( z \) e.g., (color = red) rules out some assignments of values to some of the random variables.

![Diagram showing possible worlds and evidence](image)

**Evidence redistributes probability mass over possible worlds**

A given piece of evidence \( z \) rules out all possible worlds that are incompatible with \( z \) or selects the possible worlds in which \( z \) is True. Evidence \( z \) induces a distribution \( P_z(\cdot) \):

\[
P_z(e) = \begin{cases} 
\frac{1}{P(z)} P(e) & \text{if } e \models z \\
0 & \text{if } e \nvdash z 
\end{cases}
\]

\[
P(h|z) = \sum_{e \in E_z} P_z(e) = \frac{1}{P(z)} \sum_{e \in E_z} P(e) = \frac{P(h \land z)}{P(z)}
\]

---

**Bayesian Reasoning, Classification, and Learning Classifiers from Data**

*Probability is the logic of Science (Jaynes)*

Bayesian (subjective) probability provides a basis for updating beliefs based on evidence. By updating beliefs about hypotheses based on data, we can learn about the world.

Bayesian framework provides a sound probabilistic basis for understanding many learning algorithms and designing new algorithms.

Provides practical learning algorithms:
- Naive Bayes
- Bayesian networks
- .....
Bayesian Classification

Consider the problem of classifying an instance \( X \) into one of two mutually exclusive classes \( \omega_1 \) or \( \omega_2 \)

\[
P(\omega_1 | X) = \text{probability of class } \omega_1 \text{ given the evidence } X
\]

\[
P(\omega_2 | X) = \text{probability of class } \omega_2 \text{ given the evidence } X
\]

What is the probability of error?

\[
P(error | X) = P(\omega_1 | X) \text{ if we choose } \omega_2
\]

\[
P(error | X) = P(\omega_2 | X) \text{ if we choose } \omega_1
\]

Bayesian Optimal Classification

Consider the problem of classifying an instance \( X \) into one of two mutually exclusive classes \( \omega_1 \) or \( \omega_2 \)

\[
P(\omega_1 | X) = P(X | \omega_1)P(\omega_1);
\]

\[
P(\omega_2 | X) = P(X | \omega_2)P(\omega_2)
\]

What is the probability of error?

\[
P(error | X) = P(\omega_1 | X) \text{ if we choose } \omega_2
\]

\[
P(error | X) = P(\omega_2 | X) \text{ if we choose } \omega_1
\]

Bayesian Optimal Classification

To minimize classification error

Choose \( \omega_1 \) if \( P(\omega_1 | X) > P(\omega_2 | X) \)

Choose \( \omega_2 \) if \( P(\omega_2 | X) > P(\omega_1 | X) \)

which yields

\[
P(error | X) = \min\{P(\omega_1 | X), P(\omega_2 | X)\}
\]
Bayes Optimal Classification

Classification rule that guarantees minimum error:

Choose $\omega_1$ if $P(X \mid \omega_1) P(\omega_1) > P(X \mid \omega_2) P(\omega_2)$

Choose $\omega_2$ if $P(X \mid \omega_2) P(\omega_2) > P(X \mid \omega_1) P(\omega_1)$

If $P(X \mid \omega_1) = P(X \mid \omega_2)$

classification depends entirely on $P(\omega_1)$ and $P(\omega_2)$

If $P(\omega_1) = P(\omega_2)$,
classification depends entirely on $P(X \mid \omega_1)$ and $P(X \mid \omega_2)$

Bayes classification rule combines the effect of the two terms
optimally - so as to yield minimum error classification.

Generalization to multiple classes $c(X) = \arg \max_{\omega_i} P(\omega_i \mid X)$

Minimum Risk Classification

Let $\lambda_j$ = risk or cost associated with assigning an instance
to class $\omega_j$ when the correct classification is $\omega_j$

$R(\omega_j \mid X)$ = expected loss incurred in assigning $X$ to class $\omega_j$

$R(\omega_1 \mid X) = \lambda_1 P(\omega_1 \mid X) + \lambda_2 P(\omega_2 \mid X)$

$R(\omega_2 \mid X) = \lambda_2 P(\omega_1 \mid X) + \lambda_2 P(\omega_2 \mid X)$

Classification rule that guarantees minimum risk:

Choose $\omega_1$ if $R(\omega_1 \mid X) < R(\omega_2 \mid X)$

Choose $\omega_2$ if $R(\omega_2 \mid X) < R(\omega_1 \mid X)$

Flip a coin otherwise

Minimum Risk Classification

Ordinarily $(\lambda_1 - \lambda_2)$ and $(\lambda_1 - \lambda_2)$ are positive
(cost of being correct is less than the cost of error)

So we choose $\omega_i$ if

$P(X \mid \omega_i) - \frac{\lambda_i - \lambda_j}{\lambda_i - \lambda_j} P(\omega_i)$

Minimum error classification rule is a special case:

$\lambda_i = 0 \text{ if } i = j$ and $\lambda_i = 1 \text{ if } i \neq j$

Question: How to compute $P(X \mid \omega_i)$?
Measuring classifier performance

True error of $h$, $\text{Error}_h = \Pr \{ f(x) \neq h(x) \}$

Estimated error of $h$, $\text{Error}_h^*(h) = \frac{1}{|S|} \sum_{x \in S} \delta(f(x) \neq h(x))$

$\delta(a,b) = 1$ iff $a \neq b$; $\delta(a,b) = 0$ otherwise

Evaluating the performance of the decision tree

Sample error estimated from training data is an optimistic estimate $\text{Bias} = E[\text{Error}_h]\text{Bias} = \text{Error}_h - \text{Error}_h^*(h)$

For an unbiased estimate, $h$ must be evaluated on an independent sample $S$ (which is not the case if $S$ is the training set!)

Even when the estimate is unbiased, it can vary across samples!

If $h$ misclassifies 8 out of 100 samples $\text{Error}_h = \frac{8}{100} = 0.08$

How close is the sample error to the true error?
Evaluation of a classifier with limited data

There is extensive literature on how to estimate classifier performance from samples and how to assign confidence to estimates (See Mitchell, Chapter 5)

Holdout method – use part of the data for training, and the rest for testing

We may be unlucky – training data or test data may not be representative

Solution – Run multiple experiments with disjoint training and test data sets in which each class is represented in roughly the same proportion as in the entire data set

K-fold cross-validation

Recommended procedure for evaluating classifiers when data are limited

Use K-fold cross-validation (K=5 or 10)

Better still, repeat K-fold cross-validation R times and average the results
Leave-one-out cross-validation

$K$-fold cross validation with $K = n$ where $n$ is the total number of samples available
$n$ experiments – using $n-1$ samples for training and the remaining sample for testing
Leave-one-out cross-validation does not guarantee the same class distribution in training and test data!
Extreme case: 50% class 1, 50% class 2
Predict majority class label in the training data
True error – 50%; Leave-one-out error estimate – 100%!!!!!!

Evaluating the performance of classifiers

Rigorous statistical evaluation of learned classifiers is important
How good is a learned hypothesis?
Is one hypothesis better than another?
Is one learning algorithm better than another on a particular learning task? (No learning algorithm outperforms all others on all tasks – No Free Lunch Theorem)
Different procedures for evaluation are appropriate under different conditions (large versus limited versus small sample) – Important to know when to use which evaluation method and be aware of pathological behavior (tendency to grossly overestimate or underestimate the target value under specific conditions)

Measuring classifier performance

To simplify matters, assume that class labels are binary
M-class problem is turned into M 2-class problems
Classifier Learning -- Measuring Performance

\[ N: \text{Total number of instances in the data set} \]
\[ TP_j: \text{True positives for class } j \]
\[ FP_j: \text{False positives for class } j \]
\[ TN_j: \text{True Negatives for class } j \]
\[ FN_j: \text{False Negatives for class } j \]

\[ \text{Accuracy}_j = \frac{TP_j + TN_j}{N} \]
\[ \text{Recall}_j = \frac{TP_j}{TP_j + FN_j} \]
\[ \text{Precision}_j = \frac{TP_j}{TP_j + FP_j} \]
\[ \text{FalseAlarm}_j = 1 - \text{Precision}_j \]

\[ \text{CorrelationCoeff} = \frac{[TP_j + FN_j] [FP_j + TN_j] - [TP_j + FP_j] [FN_j + TN_j]}{[TP_j + FN_j] [FP_j + TN_j]} \]

6/15/2004 Copyright Vasant Honavar, 2004

Micro averaging gives equal importance to each instance ⇒ classes with large number of instances dominate

\[ \text{(Micro)Average Precision} = \frac{\sum TP_j}{\sum TP_j + \sum FP_j} \]
\[ \text{(Micro)Average Recall} = \frac{\sum TP_j}{\sum TP_j + \sum FN_j} \]
\[ \text{(Micro)Average FalseAlarm} = 1 - \text{(Micro)Average Precision} \]

\[ \text{(Micro)Average CorrelationCoeff} = \frac{\sum [TP_j + FN_j] [FP_j + TN_j] - \sum [TP_j + FP_j] [FN_j + TN_j]}{\sum [TP_j + FN_j] [FP_j + TN_j]} \]

Macro averaging gives equal importance to class ⇒ performance on classes with few instances is weighted as much as performance on classes with many instances

\[ \text{(Macro)Average Precision} = \frac{1}{M} \sum \text{Precision}_j \]
\[ \text{(Macro)Average Recall} = \frac{1}{M} \sum \text{Recall}_j \]
\[ \text{(Macro)Average FalseAlarm} = 1 - \text{(Macro)Average Precision} \]
\[ \text{(Macro)Average CorrelationCoeff} = \frac{1}{M} \sum \text{CorrelationCoeff}_j \]

Precision is sometimes called specificity and Recall is sometimes called sensitivity

6/15/2004 Copyright Vasant Honavar, 2004
Classifier Learning -- Measuring Performance

<table>
<thead>
<tr>
<th>Guess</th>
<th>~O₁</th>
<th>O₁</th>
<th>Guess</th>
<th>~O₂</th>
<th>O₂</th>
<th>Guess</th>
<th>~O₃</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>70</td>
<td>10</td>
<td>True</td>
<td>40</td>
<td>20</td>
<td>True</td>
<td>61</td>
<td>9</td>
</tr>
<tr>
<td>C₁</td>
<td>5</td>
<td>15</td>
<td>C₂</td>
<td>14</td>
<td>26</td>
<td>C₃</td>
<td>19</td>
<td>11</td>
</tr>
</tbody>
</table>

\[
\text{precision } P_1 = \frac{15}{15 + 10} \quad \text{recall } R_1 = \frac{15}{15 + 5} \quad R_{\text{micro}} = \frac{15 + 26 + 11}{15 + 26 + 11 + 10 + 20 + 9} \\
\text{precision } P_2 = \frac{26}{26 + 20} \quad \text{precision } P_3 = \frac{11}{11 + 9} \quad \text{Accuracy} = \frac{15 + 26 + 11}{100}
\]

The contingency table consisting of \( FN, TP, FP, TN \) contains all the information needed to assess the performance of binary classifiers. Measures like Precision, Recall, Accuracy summarize this information in the form of a single scalar. Any such summary necessarily loses information. Each measure is useful in its own way, but must be used with care. For example, accuracy is misleading when data set has an uneven proportion of examples of different classes.

If a single measure of performance is to be reported, perhaps one of the least biased and the most useful measures is the Correlation Coefficient -- Value of 1 corresponds to the perfect classifier; 0 corresponds to random predictions. Correlation coefficient can be defined for the case of \( M \)-ary classifiers.

It is often possible to trade off precision against recall.

---

Learning Classifiers

Target function \( f \) -- unknown to the learner -- \( f \in F \)
Learner’s hypothesis about what \( f \) might be -- \( h \in H \)

\( H \) -- hypothesis space
Instance space -- \( X \) -- domain of \( f, h \) - instances
Output space -- \( Y \) -- range of \( f, h \) - classes
Example -- an ordered pair \((x, y)\) where
\( x \in X \) and \( f(x) = y \in Y \)

\( F \) and \( H \) may or may not be the same!
Training set \( \mathcal{E} \) -- a multi set of examples
Learning algorithm \( L \) -- a procedure which given some \( \mathcal{E} \), outputs an \( h \in H \).
Bayesian Framework for Learning

\[ P(h | D) = \frac{P(D | h) P(h)}{P(D)} \]

- \( P(h) \) = prior probability of hypothesis \( h \)
- \( P(D) \) = prior probability of training data \( D \)
- \( P(h | D) \) = probability of \( h \) given \( D \)
- \( P(D | h) \) = probability of \( D \) given \( h \)

Bayesian Recipe for learning

Choose the most likely hypothesis given the data

\[ h_{\text{MAP}} = \underset{h \in H}{\text{argmax}} P(h | D) \quad \text{(Maximum a posteriori hypothesis)} \]

\[ h_{\text{ML}} = \underset{h \in H}{\text{argmax}} \frac{P(D | h) P(h)}{P(D)} \]

\[ = \underset{h \in H}{\text{argmax}} P(D | h) P(h) \]

If \( \forall h, h_j \in H \), \( P(b_j) = P(b_j) \),

\[ h_{\text{ML}} = \underset{h \in H}{\text{argmax}} P(D | h) \quad \text{(Maximum likelihood hypothesis)} \]
Brute Force MAP Hypothesis Learner

For each hypothesis \( h \) in \( H \), calculate the posterior probability

\[
P(h | D) = \frac{P(D | h)P(h)}{P(D)}
\]

Output the hypothesis with the highest posterior probability

\[
h_{\text{MAP}} = \arg \max_{h \in H} P(D | h)P(h)
\]

\[
h_{\text{ML}} = \arg \max_{h \in H} P(D | h)
\]

Bayesian Learning of Classifiers

If the training examples are

- independently identically distributed
- noise-free and
- if each candidate hypothesis in \( H \) is equally a priori equally likely

then every hypothesis that is consistent with the training data (that is, correctly classifies each training example) maximizes \( P(h | D) \)

Thus, Bayesian learning amounts to finding a hypothesis that is consistent with the training data
Bayesian Learning of Classifiers

If the training data are noise-free but each candidate hypothesis in $H$ is not equally a priori equally likely, Bayesian learning amounts to finding a hypothesis that trades off the error on the training data against complexity of the hypothesis.

Bayesian Learning of a Real Valued Function

Consider a real-valued target function $f$. Training examples $\langle x_i, d_i \rangle$, where $d_i$ is noisy training value $d_i = f(x_i) + e_i$.

$e_i$ is random variable (noise) drawn independently for each $x_i$ according to Gaussian distribution with zero mean.

Then the maximum likelihood hypothesis $h_{ML}$ is one that minimizes the sum of squared error:

$$ h_{ML} = \arg \min_{h(x_i)} \sum \frac{(d_i - h(x_i))^2}{\sum i} $$
Bayesian Learning of Real Valued Functions

If the training examples are
- independently identically distributed
- Noise is zero mean Gaussian
- If each candidate hypothesis in $H$ is equally a priori equally likely

Maximizing $P(h|D)$ is equivalent to minimizing the mean squared error between the predicted and actual function values for the training data.

Bayesian Learning Framework – Summary

Bayesian learning framework
- Provides a sound probabilistic basis for understanding many learning algorithms and designing new algorithms
- Provides many learning algorithms that have proven useful in practice
Naïve Bayes Learner

We can classify X if we know P(X | ω_i)

How to learn P(X | ω_i)?

One solution: Assume that the random variables in X are conditionally independent given the class.

Result: Naïve Bayes classifier which performs optimally under certain assumptions

A simple, practical learning algorithm grounded in Probability Theory

When to use
- Attributes that describe instances are likely to be conditionally independent given classification
- There is not enough data to estimate all the probabilities reliably if we do not assume independence

Successful applications
- Diagnosis
- Document Classification
- Protein Function Classification
- Prediction of whether an amino acid belongs to the surface or core of the protein based on sequence characteristics

Conditional Independence

X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z:

\[ P(X | Y, Z) = P(X | Z) \]

that is, if

\[ (\forall x, y, z) P(X = x | Y = y, Z = z) = P(X = x | Z = z) \]
Conditional Independence

\[ \begin{align*}
P(\text{Thunder} = 1 | \text{Rain} = 1, \text{Lightening} = 1) &= P(\text{Thunder} = 1 | \text{Lightening} = 1) \newline &= P(\text{Thunder} = 1 | \text{Rain} = 0, \text{Lightening} = 1) \\
P(\text{Thunder} = 1 | \text{Rain} = 1, \text{Lightening} = 0) &= P(\text{Thunder} = 1 | \text{Lightening} = 0) \\
&= P(\text{Thunder} = 1 | \text{Rain} = 0, \text{Lightening} = 0) \\
P(\text{Thunder} = 0 | \text{Rain} = 1, \text{Lightening} = 1) &= P(\text{Thunder} = 0 | \text{Lightening} = 1) \\
&= P(\text{Thunder} = 0 | \text{Rain} = 0, \text{Lightening} = 1) \\
P(\text{Thunder} = 0 | \text{Rain} = 1, \text{Lightening} = 0) &= P(\text{Thunder} = 0 | \text{Lightening} = 0) \\
&= P(\text{Thunder} = 0 | \text{Rain} = 0, \text{Lightening} = 0)
\end{align*} \]

Conditional Independence

Let \( Z_1, \ldots, Z_n \) and \( W \) be random variables on a given event space. \( Z_1, \ldots, Z_n \) are mutually independent given \( W \) if

\[ P(Z_1, Z_2, \ldots, Z_n | W) = \prod_{i=1}^{n} P(Z_i | W) \]

\[ P(Z_i | Z_j, W) = P(Z_i | W) \text{ if } Z_i \text{ and } Z_j \text{ are independent.} \]

Note that these represent sets of equations, for all possible value assignments to random variables

Independence and Conditional Independence

Let \( Z_1, \ldots, Z_n \) and \( W \) be pairwise disjoint sets of random variables on a given event space. \( Z_1, \ldots, Z_n \) are mutually independent given \( W \) if

\[ P(Z_1 \cup \ldots \cup Z_n | W) = \prod_{i=1}^{n} P(Z_i | W) \]

\[ P(Z_i | Z_j \cup W) = P(Z_i | W) \text{ if } Z_i \text{ and } Z_j \text{ are independent.} \]

Note that these represent sets of equations, for all possible value assignments to random variables
Independence Properties of Random Variables

Let $W, X, Y, Z$ be pairwise disjoint sets of random variables on a given event space. Let $I(X, Y, Z)$ denote that $X$ and $Z$ are independent given $Y$. That is, $P(W \cup Y | Z) = P(W | Y) \cdot P(Z | Y)$ or $P(X | Y \cup Z) = P(X | Y)$ Then:

a. $I(X, Z, Y) \Rightarrow I(Y, Z, X)$

b. $I(X, Z, Y \cup W) \Rightarrow I(X, Z, Y)$

c. $I(X, Z, Y \cup W) \Rightarrow I(X, Z \cup W, Y)$

d. $I(X, Z, Y) \land I(X, Z \cup Y, W) \Rightarrow I(X, Z, Y \cup W)$

Proof: Follows from definition of independence.

Implications of Independence

Suppose we have 5 Binary features and a binary class label

Without independence, in order to specify the joint distribution, we need to specify a probability for each possible assignment of values to each variable resulting in a table of size $2^6 = 64$

Suppose the features are independent given the class label – we only need $5(2 \times 2) = 20$ entries

Naive Bayes Classifier

Consider a discrete valued target function $f : \chi \rightarrow \Omega$ where an instance $X = (X_1, X_2, ..., X_n) \in \chi$ is described in terms of attribute values $X_1 = a_1, X_2 = a_2, ..., X_n = a_n$

$\omega_{\text{MAP}} = \arg \max_{\omega \in \Omega} P(\omega) \cdot P(X_1 = a_1, X_2 = a_2, ..., X_n = a_n | \omega)$

$= \arg \max_{\omega \in \Omega} P(X_1 = a_1, X_2 = a_2, ..., X_n = a_n | \omega) P(\omega)$

$= \arg \max_{\omega \in \Omega} P(X_1 = a_1, X_2 = a_2, ..., X_n = a_n | \omega) P(\omega)$
Naive Bayes Classifier

\[ \omega_{\text{MAP}} = \arg \max_{\omega \in \Omega} P(\omega | X_1 = a_1, X_2 = a_2, \ldots, X_n = a_n) \]

\[ = \arg \max_{\omega \in \Omega} P(X_1 = a_1, X_2 = a_2, \ldots, X_n = a_n | \omega) P(\omega) \]

If the attributes are independent, we have

\[ \omega_{\text{MAP}} = \arg \max_{\omega \in \Omega} \prod_{i=1}^{n} P(X_i = a_i | \omega) P(\omega) \]

\[ = \omega_{\text{MAP}} \]

\[ = \arg \max_{\omega \in \Omega} P(\omega) \prod_{i=1}^{n} P(X_i = a_i | \omega) \]

Naive Bayes Learner

For each class \( \omega_j \in \Omega \),

\[ \hat{P}(\omega_j) \leftarrow \text{Estimate}(P(\omega_j), D) \]

For each possible value \( a_{ij} \) of \( X_i \)

\[ \hat{P}(X_i = a_{ij} | \omega_j) \leftarrow \text{Estimate}(P(a_{ij} | \omega_j), D) \]

Classify a new instance \( X \)

\[ c(X) = \arg \max_{\omega \in \Omega} \hat{P}(\omega) \prod_{i=1}^{n} \hat{P}(X_i = a_i | \omega) \]

Estimate is a procedure for estimating the relevant probabilities from data

Estimation of Probabilities from Small Samples

\[ \hat{P}(X_i = a_{ij} | \omega_j) = 0 \rightarrow \hat{P}(\omega_j) \prod_{i=1}^{n} \hat{P}(X_i = a_{ij} | \omega_j) = 0 \]

Thus, if one of the attribute values has estimated class conditional probability of 0, it dominates all other attribute values

When we have few examples, this is more likely

Solution – Use Occam’s razor

– Assume uniform distribution unless data tell us otherwise
Estimation of Probabilities from Small Samples

\[ \hat{P}(X_i = a_i | \omega_j) \leftarrow \frac{n_{i\omega_j} + mp}{n_j + m} \]

- \( n_j \) is the number of training examples of class \( \omega_j \)
- \( n_{i\omega_j} \) is the number of training examples of class \( \omega_j \)
  - which have attribute value \( a_i \) for attribute \( X_i \)
- \( p \) is the prior estimate for \( \hat{P}(X_i = a_i | \omega_j) \)
- \( m \) is the weight given to the prior

As \( n \to \infty \),
\[ \hat{P}(X_i = a_i | \omega_j) \to \frac{n_{i\omega_j}}{n_j} \]

---

Sample Applications

- Learn which news articles are of interest.
- Learn to classify web pages by topic.
- Learn to classify SPAM
- Learn to assign proteins to functional families based on amino acid composition

Naive Bayes is among the most useful algorithms

What attributes shall we use to represent text?

Learning to Classify Text

Target concept Interesting? : Documents \( \rightarrow \{+,-\} \)

Represent each document by vector of words –
- one attribute per word position in document

Value of the attribute is the word appearing in that position in the document \( d \)

Learning: Use training examples to estimate
\[ P(+), P(-), P(d | +), P(d | -) \]

\[ P(d | \omega_j) = \prod_{i=1}^{\text{length}(d)} P(X_i = a_i | \omega_j) \]
Learning to Classify Text

\[ P(d \mid \omega_j) = \prod_{i=1}^{\text{length}(d)} P(X_i \mid \omega_j) \]

This would require estimating for each document, length(d) \times \text{Vocabulary size} \times 2 probabilities. To simplify matters, assume that probability of encountering a specific word in a particular position is independent of the position. Treat each document as a bag of words!

Bag of Words Representation

So we estimate one position-independent class-conditional probability \( P(w_j \mid \omega_j) \) for each word instead of the set of probabilities \( P(X_1 = w_j \mid \omega_j) \ldots P(X_{\text{length}(d)} = w_j \mid \omega_j) \).

The number of probabilities to be estimated drops to \( |\text{Vocabulary}| \times 1 \).

Learning to Classify Text

With the bag of words representation, we have

\[ P(w_j \mid \omega_j) = \frac{n_{j,k} + 1}{n_j + |\text{Vocabulary}|} \]

\( n_j \) = number of word positions in documents of class \( \omega_j \)
\( n_{j,k} \) = number of times the word \( w_j \) appears among documents of class \( \omega_j \)

\( (m-\text{estimate with uniform priors and } m = |\text{Vocabulary}|) \)
Naïve Bayes Text Classifier

Given 1000 training documents from each group, learn to classify new documents according to the newsgroup where it belongs. Naïve Bayes achieves 89% classification accuracy.

comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.windows.x
alt.atheism
soc.religion.christian
talk.politics.misc
talk.politics.mideast
talk.politics.misc
talk.politics.queer

Naïve Bayes Text Classifier

Article from rec.sport.hockey

Path: cantaloupe.rr.cs.cmu.edu!das-news.harvard.edu!logics@um.edu
From: xxx@yyy.zzz.edu (John Doe)
Subject: Re: This year's biggest and worst (opinion)...
Date: 5 Apr 93 09:53:39 GMT

I can only comment on the Kings, but the most obvious candidate for pleasant surprise is Alex Zhitnik. He came highly touted as a defensive defenseman, but he's clearly much more than that. Great skater and hard shot (though wish he were more accurate). In fact, he pretty much allowed the Kings to trade away that huge defensive liability Paul Coffey. Kelly Hrudey is only the biggest disappointment if you thought he was any good to begin with. But, at best, he's only a mediocre goaltender. A better choice would be Tomas Sandstrom, though not through any fault of his own, but because some thugs in Toronto decided —.

Naïve Bayes Learner – Summary

Produces minimum error classifier if attributes are conditionally independent given the class.

When to use
  - Attributes that describe instances are likely to be conditionally independent given classification
  - There is not enough data to estimate all the probabilities reliably if we do not assume independence

Often works well even if when independence assumption is violated (Domigos and Pazzani, 1996)
Naive Bayes assumption of conditional independence can be too restrictive.
But representing joint distributions is intractable without some independence assumptions.
Bayesian networks capture conditional independence among subsets of variables.
This allows combining prior knowledge about (in)dependencies among variables with observed training data.

Bayesian Networks

Bayesian network

- Bayesian network is a directed acyclic graph (DAG) in which the nodes represent random variables.
- Each node is annotated with a probability distribution $P(X_i | Parent(X_i))$ representing the dependency of that node on its parents in the DAG.
- Each node is asserted to be conditionally independent of its non-descendants, given its immediate predecessors.
- Arcs represent direct dependencies.
Bayesian network

- Bayesian network is a directed acyclic graph (DAG) in which the nodes represent random variables.
- Each node is annotated with a probability distribution $P(X_i / \text{Parents}(X_i))$ representing the dependency of that node on its parents in the DAG.
- Each node is asserted to be conditionally independent of its non-descendants, given its immediate predecessors.
- Arcs represent direct dependencies.

Bayesian Networks

$$P(X_i | X_{i-1}, \ldots, X_1) = \prod_{j=1}^{i} P(X_j | X_{j-1}, \ldots, X_1) = \prod_{j=1}^{i} P(X_j | \text{Parents}(X_j))$$

Efficient factorized representation of probability distributions via conditional independence.

Bayesian Networks

- Qualitative part: statistical independence statements.
- Directed acyclic graph (DAG):
  - Nodes - random variables.
  - Edges - direct influence.
- Quantitative part: Conditional probability distributions - one for each random variable conditioned on its parents.
Qualitative part

Nodes are independent of non-descendants given their parents
- \( P(R|E=y,A) = P(R|E=y) \) for all values of \( R,A,E \)

Given that there is an earthquake,
I can predict a radio announcement regardless of whether the alarm sounds
- \( d \)-separation: a graph theoretic criterion for reading independence statements
Can be computed in linear time (on the number of edges)

Quantitative Part

Associated with each node \( X_i \) there is a set of conditional probability distributions \( P(X_i|P_{ai};\Theta) \)
- If variables are discrete, \( \Theta \) is usually multinomial

- Variables can be continuous, \( \Theta \) can be Gaussian
- Combinations of discrete and continuous are only constrained by available inference mechanisms

Bayesian Networks: Summary

Bayesian networks extend the idea of exploiting independence beyond Naïve Bayes to factored representations of more complex probability distributions

Learning Bayesian networks from data
- Learning the structure
- Given a structure, learning the parameters

Both problems can be approached from a Bayesian perspective but we have not covered the details of inference and learning using general Bayesian Networks
Modeling dependencies among input variables using Decision Tree Classifiers

- Decision tree Representation
- Elements of information theory
- How to learn decision trees from data
- Over-fitting and how to minimize it
- How to deal with missing values in the data
- Recent results
  - Learning decision trees from distributed data
  - Learning decision trees at multiple levels of abstraction

Decision tree representation

In the simplest case,
- each internal node tests on an attribute
- each branch corresponds to an attribute value
- each leaf node corresponds to a class label

In general,
- each internal node corresponds to a test (on input instances) with mutually exclusive and exhaustive outcomes – tests may be univariate or multivariate
- each branch corresponds to an outcome of a test
- each leaf node corresponds to a class label
Decision tree representation

Any Boolean function can be represented by a decision tree:

Any function \( f : A_1 \times A_2 \times \cdots \times A_n \rightarrow C \)

where each \( A_i \) is the domain of the \( i \)th attribute and \( C \) is a discrete set of values (class labels) can be represented by a decision tree.

In general, the inputs need not be discrete valued.

Learning Decision Tree Classifiers

Decision trees are especially well suited for representing simple rules for classifying instances that are described by discrete attribute values.

Decision tree learning algorithms
- Implement Ockham's razor as a preference bias (simpler decision trees are preferred over more complex trees)
- Are relatively efficient – linear in the size of the decision tree and the size of the data set
- Produce comprehensible results
- Are often among the first to be tried on a new data set
Learning Decision Tree Classifiers

Ockham’s razor recommends that we pick the simplest decision tree that is consistent with the training set. The simplest tree is one that takes the fewest bits to encode (why? – information theory). There are far too many trees that are consistent with the training set. Searching for the simplest tree that is consistent with the training set is not typically computationally feasible.

Solution

– use a greedy algorithm – not guaranteed to find the simplest tree – but works well in practice
– or restrict the space of hypothesis to a subset of simple trees

Information – Some intuitions

Information reduces uncertainty.
Information is relative – to what you already know.
Information is related to surprise.
Information depends on context.

Digression: Information and Uncertainty

You are stuck inside. You send me out to report back to you on what the weather is like. I do not lie, so you trust me. You and I are both generally familiar with the weather in Iowa.

On a July afternoon in Iowa, I walk into the room and tell you it is hot outside.
On a December afternoon in Iowa, I walk into the room and tell you it is hot outside.
Digression: Information and Uncertainty

How much information does a message contain?

If my message to you describes a scenario that you expect with certainty, the information content of the message for you is zero.

The more surprising the message to the receiver, the greater the amount of information conveyed by the message.

What does it mean for a message to be surprising?

Suppose I have a coin with heads on both sides and you know that I have a coin with heads on both sides.

I toss the coin, and without showing you the outcome, tell you that it came up heads. How much information did I give you?

Suppose I have a fair coin and you know that I have a fair coin.

I toss the coin, and without showing you the outcome, tell you that it came up heads. How much information did I give you?

Information

Without loss of generality, assume that messages are binary – made of 0s and 1s.

Conveying the outcome of a fair coin toss requires 1 bit of information – need to identify one out of two equally likely outcomes.

Conveying the outcome one of an experiment with 8 equally likely outcomes requires 3 bits.

Conveying an outcome of that is certain takes 0 bits.

In general, if an outcome has a probability \( p \), the information content of the corresponding message is

\[
I(p) = -\log_2 p \\
I(0) = 0
\]
**Subjective nature of information**

Suppose there are 3 agents – Adrian, Anna, Jun, in a world where a dice has been tossed. Adrian observes the outcome is a “6” and whispers to Anna that the outcome is “even” but Jun knows nothing about the outcome.

Probability assigned by Anna to the event “6” is a subjective measure of Anna’s belief about the state of the world.

Information gained by Adrian by looking at the outcome of the dice = log₂6 bits.

Information conveyed by Adrian to Anna = log₂6 – log₂3 bits

Information conveyed by Adrian to Jun = 0 bits

---

**Information and Shannon Entropy**

Suppose we have a message that conveys the result of a random experiment with \( m \) possible discrete outcomes, with probabilities \( p_1, p_2, \ldots, p_m \).

The expected information content of such a message is called the entropy of the probability distribution

\[
\mathbb{E}(p_1, p_2, \ldots, p_m) = \sum_{i=1}^{m} p_i I(p_i)
\]

\[
I(p_i) = -\log_2 p_i \quad \text{provided } p_i \neq 0
\]

\[
I(p_i) = 0 \quad \text{otherwise}
\]

---

**Shannon’s entropy as a measure of information**

Let \( \mathbb{P} = (p_1, \ldots, p_n) \) be a discrete probability distribution.

The entropy of the distribution \( \mathbb{P} \) is given by

\[
H(\mathbb{P}) = -\sum_{i=1}^{n} p_i \log_2(p_i)
\]

\[
H\left(\frac{1}{2}, \frac{1}{2}\right) = -\sum_{i=1}^{2} \frac{1}{2} \log_2\left(\frac{1}{2}\right) = \frac{1}{2} \log_2\left(\frac{1}{2}\right) + \frac{1}{2} \log_2\left(\frac{1}{2}\right) = 1 \text{ bit}
\]

\[
H(0,1) = -\sum_{i=1}^{2} p_i \log_2(p_i) = -0(1) - 1(0) = 0 \text{ bit}
\]
Shannon’s entropy as a measure of information

For any distribution \( P \), \( H(P) \) is the optimal number of binary questions required on average to determine an outcome drawn from \( P \).

We can extend these ideas to talk about how much information is conveyed by the observation of the outcome of one experiment about the possible outcomes of another (mutual information).

We can also quantify the difference between two probability distributions (Kullback-Liebler divergence or relative entropy).

Coding Theory Perspective

Suppose you and I both know the distribution \( P \).

I choose an outcome according to \( P \).

Suppose I want to send you a message about the outcome.

You and I could agree in advance on the questions I can simply send you the answers.

Optimal message length on average is \( H(P) \).

Generalizes to noisy communication.
Learning Decision Tree Classifiers

On average, the information needed to convey the class membership of a random instance from \( S = E(S) \)

\[
\begin{align*}
\text{Nature} & \rightarrow \text{Training Data } S_1, S_2, S_3 \\
\text{Instance} & \rightarrow \text{Classifier} \\
\text{Class label} & \\
\end{align*}
\]

\[ E(S) = -\sum_{i=1}^{n} \left( \frac{|E_i|}{|F|} \right) \log_2 \left( \frac{|E_i|}{|F|} \right) \]

\( S_i \) is the multi-set of examples belonging to class \( C_i \)

---

The task of the learner then is to extract the needed information from the training set and store it in the form of a decision tree for classification.

Information gain based decision tree learner:

- Start with the entire training set at the root.
- Recursively add nodes to the tree corresponding to tests that yield the greatest expected reduction in entropy (or the largest expected information gain).
- Until some termination criterion is met (e.g., the training data at every leaf node has zero entropy).

---

Learning Decision Tree Classifiers - Example

Training Data

Instances – ordered 3-tuples of attribute values corresponding to
- Height (tall, short)
- Hair (dark, blonde, red)
- Eye (blue, brown)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Height</th>
<th>Hair</th>
<th>Eye</th>
<th>Class label</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_1</td>
<td>t</td>
<td>d</td>
<td>l</td>
<td>+</td>
</tr>
<tr>
<td>I_2</td>
<td>s</td>
<td>d</td>
<td>l</td>
<td>+</td>
</tr>
<tr>
<td>I_3</td>
<td>t</td>
<td>b</td>
<td>l</td>
<td>+</td>
</tr>
<tr>
<td>I_4</td>
<td>t</td>
<td>r</td>
<td>l</td>
<td>+</td>
</tr>
<tr>
<td>I_5</td>
<td>s</td>
<td>b</td>
<td>l</td>
<td>+</td>
</tr>
<tr>
<td>I_6</td>
<td>t</td>
<td>b</td>
<td>w</td>
<td>+</td>
</tr>
<tr>
<td>I_7</td>
<td>t</td>
<td>d</td>
<td>w</td>
<td>+</td>
</tr>
<tr>
<td>I_8</td>
<td>s</td>
<td>b</td>
<td>w</td>
<td>+</td>
</tr>
</tbody>
</table>
Learning Decision Tree Classifiers - Example

Hair is the most informative because it yields the largest reduction in entropy. Test on the value of Hair is chosen to correspond to the root of the decision tree.

Similarly, we have:

\[ E(S; \text{Hair}) = \frac{1}{2} E(S_1; \text{Hair}) + \frac{1}{2} E(S_2; \text{Hair}) = 0.607\text{bits} \]

Consider the error of a hypothesis \( h \) over

- training data: \( \text{Error}_{\text{train}}(h) \)
- entire distribution \( D \) of data: \( \text{Error}_{D}(h) \)

Hypothesis \( h \in H \) over fits training data if there is an alternative hypothesis \( h' \in H \) such that

\[ \text{Error}_{\text{train}}(h) < \text{Error}_{\text{train}}(h') \]

\[ \text{Error}_{D}(h) > \text{Error}_{D}(h') \]
Causes of over fitting

As we move further away from the root, the data set used to choose the best test becomes smaller → poor estimates of entropy

Noisy examples can further exacerbate over fitting

Minimizing over fitting

- Use roughly the same size sample at every node to estimate entropy → when there is a large data set from which we can sample
- Stop when further split fails to yield statistically significant information gain (estimated from validation set)
- Grow full tree, then prune
  - minimize size (tree) + size (exceptions (tree))
Reduced error pruning

Each decision node in the tree is considered as a candidate for pruning.

Pruning a decision node consists of:
- removing the sub tree rooted at that node,
- making it a leaf node, and
- assigning it the most common label at that node.

### Reduced error pruning – Example

<table>
<thead>
<tr>
<th>Node</th>
<th>Accuracy gain by Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-20%</td>
</tr>
<tr>
<td>B</td>
<td>+10%</td>
</tr>
</tbody>
</table>

Before Pruning

After Pruning

Reduced error pruning

Do until further pruning is harmful:
- Evaluate impact on validation set of pruning each candidate node.
- Greedily select a node which most improves the performance on the validation set when the sub tree rooted at that node is pruned.

Drawback – holding back the validation set limits the amount of training data available – not desirable when data set is small.
Pruning based on whether information gain is significantly greater than zero

Evaluate Candidate split to decide if the resulting information gain is significantly greater than zero as determined using a suitable hypothesis testing method at a desired significance level.

Example: \( \chi^2 \) statistic

In the 2-class, binary (L,R) split case, \( n_i \) of class 1, \( n_j \) of class 2: \( \chi^2 = \sum_{i=1}^{\min(n_i, n_j)} \frac{(n_i - n_j)^2}{n_i} \)

Split sends \( pN \) to L and \( (1-p)N \) to R.

Random split would send \( pN \) of class 1 to L and \( n_j \) of class 2 to L.

The critical value of \( \chi^2 \) depends on the degrees of freedom which is 1 in this case (for a given \( p \), \( n_j \) fully specifies \( n_{1L}, n_{1R} \) and \( n_{2L} \)).

In general, the number of degrees of freedom can be > 1.
Pruning based on whether information gain is significantly greater than zero

\[ \chi^2 = \sum_{i=1}^{\text{Branches}} \sum_{j=1}^{\text{Classes}} \frac{(n_{ij} - n_{i.})^2}{n_{i.}} \]

The greater the value of \( \chi^2 \), the less likely it is that the split is random. For a sufficiently high value of \( \chi^2 \), the difference between the expected (random) split is statistically significant and we reject the null hypothesis that the split is random.

\[ N = n_1 + n_2 + \ldots + n_{\text{Classes}} \]
\[ p = [p_1, p_2, \ldots, p_{\text{Branches}}]; \sum_{j=1}^{\text{Branches}} p_j = 1 \]
\[ n_{i.} = p_i n \]

Degrees of freedom = (\( \text{Classes} - 1 \)) (\( \text{Branches} - 1 \))

---

Rule post-pruning

Convert tree to equivalent set of rules

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others by dropping a condition at a time if doing so does not reduce estimated accuracy (at the desired confidence level)
3. Sort final rules in order of lowest to highest error

Advantage – can potentially correct bad choices made close to the root

Post pruning based on validation set is the most commonly used method in practice

Development of pre pruning methods with comparable performance that do not require a validation set is an open problem
**Classification of instances**

- Unique classification – possible when each leaf has zero entropy and there are no missing attribute values.
- Most likely classification – based on distribution of classes at a node when there are no missing attribute values.
- Probabilistic classification – based on distribution of classes at a node when there are no missing attribute values.

**Handling different types of attribute values**

Types of attributes:
- Nominal – values are names.
- Ordinal – values are ordered.
- Cardinal (Numeric) – values are numbers (hence ordered).

**Handling numeric attributes**

<table>
<thead>
<tr>
<th>Attribute T</th>
<th>40</th>
<th>48</th>
<th>50</th>
<th>54</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Candidate splits:
- \( T > \frac{(48 + 50)}{2} \)?
- \( T > \frac{(60 + 70)}{2} \)?

\[ E(S \mid T > 49?) = \frac{2}{6} (0) + \frac{4}{6} \left( \frac{3}{4} \log \left( \frac{1}{4} \right) + \frac{1}{4} \log \left( \frac{1}{4} \right) \right) \]

Sort instances by value of numeric attribute under consideration.
For each attribute, find the test which yields the lowest entropy.
Greedily choose the best test across all attributes.
Handling numeric attributes

**Axis-parallel split**

![Axis-parallel split diagram]

**Oblique split**

![Oblique split diagram]

Oblique splits cannot be realized by univariate tests

Two-way versus multi-way splits

Entropy criterion favors many-valued attributes
Pathological behavior – what if in a medical diagnosis data set, social security number is one of the candidate attributes?

Solutions

Only two-way splits (CART) \( A = \text{value} \) versus \( A = \neg \text{value} \)

Entropy ratio \((C4.5)\)

\[
\text{Entropy Ratio}(S \mid A) = \frac{\text{Entropy}(S \mid A)}{\text{Split Entropy}(S \mid A)}
\]

\[
\text{Split Entropy}(S \mid A) = -\sum_{i} \left| S_i \right| \log \left( \frac{\left| S_i \right|}{\left| S \right|} \right)
\]

Alternative split criteria

Consider split of set \( S \) based on attribute \( A \)

\[
\text{Impurity}(S \mid A) = \sum_{j=1}^{\text{Class}(A)} \text{Impurity}(S_j)
\]

Entropy

\[
\text{Impurity}(Z) = \sum_{i} \left( \frac{Z_i}{|Z|} \log \frac{Z_i}{|Z|} \right)
\]

Gini

\[
\text{Impurity}(Z) = \sum_{i} \left( \frac{Z_i}{|Z|} \right) \left( 1 - \frac{Z_i}{|Z|} \right)^2
\]

(Expected rate of error if class label is picked randomly according to distribution of instances in a set)
Alternative split criteria

One-sided split criteria – often useful for exploratory analysis of data

\[ \text{Impurity}(S | A) = \min_{i \in \text{Values}(A)} \{ \text{Impurity}(S_i) \} \]

Incorporating Attribute costs

Not all attribute measurements are equally costly or risky

In Medical diagnosis

Blood-Test has cost $150
Exploratory-Surgery may have a cost of $3000

Goal: Learn a Decision Tree Classifier which minimizes cost of classification

Tan and Schlimmer (1990)
Nunez (1988)

\[ \frac{\text{Gain}^2(S, A)}{\text{Cost}(A)} \]

where \( w \in [0, 1] \) determines importance of cost

Incorporating Different Misclassification Costs for different classes

Not all misclassifications are equally costly
An occasional false alarm about a nuclear power plant meltdown is less costly than the failure to alert when there is a chance of a meltdown

Weighted Gini Impurity

\[ \text{Impurity}(S) = \sum_{i} \lambda_{ij} \left( \frac{|S_i|}{|S|} \right) \left( \frac{|S_j|}{|S|} \right) \]

\( \lambda_{ij} \) is the cost of wrongly assigning an instance belonging to class \( i \) to class \( j \)
Dealing with Missing Attribute Values (Solution 1)

Sometimes, the fact that an attribute value is missing might itself be informative –
Missing blood sugar level might imply that the physician had reason not to measure it
Introduce a new value (one per attribute) to denote a missing value
Decision tree construction and use of tree for classification proceed as before

Dealing with Missing Attribute Values (Solution 2)

During decision tree construction
Generate several fractionally weighted training examples based on the distribution of values for the corresponding attribute at the node

During use of tree for classification
Generate multiple instances by assigning candidate values for the missing attribute based on the distribution of instances at the node
Sort each such instance through the tree to generate candidate labels and assign the most probable class label or probabilistically assign class label

Dealing with Missing Attribute Values

Suppose B is missing

Fractional instance based on distribution at the node for class + .. 4/5 for B=1, 1/5 for B=0
Fractional instance based on distribution at the node for class + .. 1/5 for B=0, 0 for B=1

n+=60, n-=40
A
B

(n|A=0) = 10; (n|A=0) = 40

(n|A=1)=50

(n|A=0, B=1)=40

(n|A=0, B=0) = 10
Summary of Decision Trees

Simple
Fast (Linear in size of the tree, linear in the size of the training set, linear in the number of attributes)
Produce easy to interpret rules
Good for generating simple predictive rules from data with lots of attributes

Additional Information

- Learning Decision Trees from Distributed Data (Caragea, Silvescu and Honavar, 2001; 2004)
- Learning Decision Trees from Attribute Value Taxonomies and partially specified data (Zhang, Silvescu and Honavar, 2002; Zhang and Honavar, 2003)
- Learning Decision Trees from Relational Databases (Atramentov, Leiva and Honavar, 2003)
- Motif-Based Decision Trees for Protein function classification (Wang, Schroeder, Dobbs, and Honavar, 2003)
Computational Biology Application I

Protein Function Prediction from amino acid sequence
Can we use machine learning algorithms to discover sequence correlates of functionally significant structural features of proteins?

Protein Function Prediction: Background

Why are proteins important?
- Enzymes - catalyze biochemical reactions
- Transport proteins – Hemoglobin and Myoglobin transport oxygen
- Membrane proteins – e.g., Potassium channels
- Structural proteins – hair, bone, etc.
- Defense proteins – e.g., Immunoglobulins
- Regulatory proteins – e.g., Hormones

Protein Function Prediction: Background

Multiple facets of protein function [Eisenberg et al., 2000]
- Enzymatic function or biochemical function
- Molecular function – defined in terms of interactions with other proteins or ligands
- Cellular function – defined by cellular localization and role in specific signaling or metabolic pathways
- Phenotypic function – defined in terms of effects of mutation or deletion of the gene that codes for the protein

There is a need to analyze diverse sources of data about protein function from different perspectives
Protein Function Prediction: Background

- There are about 80,000 to 100,000 genes in the human genome
- There are about 3,000 to 4,000 proteins in a typical cell
- There are about 19,000 protein structures in the Protein Databank
- There are over 1,000,000 protein sequences in SWISS-PROT
- Experimental determination of protein structure lags far behind!

Protein Function Prediction: Background

Information Flow

DNA → RNA → Protein
- Protein sequence largely determines protein 3-dimensional structure
- Protein structure largely determines protein function
- Similar sequences can result in different structures
- Dissimilar sequences can yield similar structures
- Many proteins are multi-functional

Protein Function Prediction: Approaches

- Sequence homology based methods
- Characteristic motif based methods
- Structure based methods (structure prediction using homology modeling or ab initio structure prediction using molecular dynamics, followed by function prediction)
- Data-driven discovery of sequence based features or combinations of features that are predictive of protein function using machine learning
Synthesis of Protein Function Classifiers

- Short amino acid sequence patterns are conserved within families of closely related proteins
- A variety of algorithms have been developed for discovery of such patterns – motifs, profiles, etc.
- Each protein family contains several motifs
- A given motif typically shows up in multiple families

Can protein functional families be characterized in terms of relationships among conserved sequence patterns?
Sources of Sequence Motifs Used

- PROSITE patterns and profiles [Hofmann, et al., 1999] – expert curated database which identifies characteristic motif(s) for each functional family; Motif composition of a sequence found using PROFILES CAN

- MEME [Bailey et al., 1999] which identifies conserved motifs from a given set of sequences using multiple sequence alignment; Motif composition of a sequence found using MAST

Decision Trees for MEROPS Families

Peptidase (protease) – proteins that cause the hydrolysis (cleavage) of peptide bonds

MEROPS – A Two-level classification system [Rawlings et al., 2000]
- Family - protein sequences with statistically significant sequence similarities.
- Clan - families with common evolutionary origins (have similar tertiary folds or the order of catalytic-site residues).

Sequence similarity within clans is low (less than 30%)

Data: 1933 proteins consisting of 84 MEROPS Peptidase families. 3 to 313 proteins per family
- 19 MEROPS Peptidase clans. 1 to 18 families per clan
- 1627 non-redundant sequences selected using PURGE
### Examples of clans and families

<table>
<thead>
<tr>
<th>Clan</th>
<th>Family</th>
<th>Representative enzyme</th>
<th>3D</th>
<th>PROSITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA</td>
<td>S1</td>
<td>Chymotrypsin / trypsin</td>
<td>Yes</td>
<td>PDO000004</td>
</tr>
<tr>
<td>S2A</td>
<td></td>
<td>Alpha-Lytic endopeptidase</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>S2B</td>
<td></td>
<td>Glutamyl endopeptidase (VII) (Staphyllococcus)</td>
<td>Yes</td>
<td>PDO000571</td>
</tr>
<tr>
<td>S2C</td>
<td></td>
<td>Protease Do (htrA) (Escherichia)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>S29</td>
<td></td>
<td>Hepatitis C virus NS3 endopeptidase</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>S30</td>
<td></td>
<td>Tobacco etch virus 35 kDa endopeptidase</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>S31</td>
<td></td>
<td>Cattle diarrhea virus p80 endopeptidase</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>S45</td>
<td>Penicillin amidohydrolase</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Decision Trees for MEROPS Families

**Method**

Generate decision trees (and corresponding rule sets) for clans and families using MEME motifs and PROSITE motifs (MEME max motif length set to 12)

Compare performance of the rule sets with that of the best single motif for each family (or clan) with respect to standard precision and recall measures using 5-fold cross-validation

### Performance Measures

- **N**: Total number of instances in the data set.
- **TP(c)**: True positive for class c
- **FP(c)**: False positive for class c
- **TN(c)**: True Negative for class c
- **FN(c)**: False Negative for class c
- **TP**: True positives over all classes
- **Accuracy**: \( \frac{TP}{N} \)
- **Precision (c)**: \( \frac{TP(c)}{TP(c) + FP(c)} \)
  - the degree to which the classifier picks out members of the target class while rejecting all others
- **Recall (c)**: \( \frac{TP(c)}{TP(c) + FN(c)} \)
  - the extent to which the classifier identifies all members of the target class
- **Best motif (for each class)**: motif in rule set with largest \( \text{Precision} \times \text{Recall} \)
Table 1: Comparison of classification performance of family and clan rules based on Meme motifs with that of single best motifs, i.e., motifs with the largest (precision × recall). Column 1 shows the average number of motifs per rule. The percentage precision and recall figures for family (clan) correspond to averages taken over families (clans). Percentage accuracy is computed over the entire test sample. All of the results represent estimates based on 5-fold cross-validation.

Table 2: Comparison of classification performance of family and clan rules based on PROSITE motifs with that of single best motifs, i.e., motifs with the largest (precision × recall). Column 1 shows the average number of motifs (checked for presence or absence) per rule. The percentage precision and recall figures for family (clan) correspond to averages taken over families (clans). Percentage accuracy is computed over the entire test sample. All of the results represent estimates based on 5-fold cross-validation.
Decision Trees Identify Combinations of Sequence Motifs Correlated with Functionally Important Structural Features

Figure 3a: The 3-dimensional structure of human Caspase-1 (MEROPS family C14), corresponding to PDB entry 1BMQ. The four labeled residues Arg 179, His 237, Cys 285, and Arg 341 are known to form the substrate binding pocket of the Caspase-1 enzyme [Wilson, et al., 1994 Nature 370:270-275]. Three of these residues (Arg 179, His 237, and Cys 285) are located within the MEME-generated motifs frequently used by the decision tree classifier for the MEROPS family C14. These motifs correspond to residues 179-190 (red), 228-239 (yellow), and 276-287 (green).

Figure 3b: The 3-dimensional structure of Astacin (MEROPS family M12) from A. astacus, corresponding to PDB entry 1QJJ. Five MEME-generated motifs selected by the decision tree algorithm for the MEROPS family M12 correspond to residues 83-94 (red), 96-107 (yellow), and 142-153 (green). The five labeled residues -- His 92, His 96, Glu 93, His 102, and Tyr 149 -- appear within the motifs and have been shown to form the zinc binding pocket of the enzyme [Bond and Beynon, 1995, Protein Science 4:1247-1261].

Decision Trees for MEROPS Families

Protein function classifiers built using motifs extracted by automated motif discovery programs like MEME compare favorably with those based on PROSITE motifs.

Fully automated data-driven approaches based on machine learning algorithms might be able to identify sequence correlates of functionally significant structural elements.

Similar studies in our lab have demonstrated the effectiveness of reduced alphabet representations of protein sequences for protein function prediction (data not shown).
Neural Network Classifiers and Support Vector Machines

Outline
- Background
- Threshold logic functions
- Connection to logic
- Connection to geometry
- Learning threshold functions – perceptron algorithm and its variants
- Perceptron convergence theorem

Background – Neural computation
- 1900 – Birth of neuroscience – Ramon Cajal et al.
- 1913 – Behaviorist or stimulus response psychology
- 1930-50: Theory of Computation, Church-Turing Thesis
- 1943: McCulloch & Pitts “A logical calculus of neuronal activity”
- 1949: Hebb – Organization of Behavior
- 1960-65: Perceptron model developed by Rosenblatt

Background – Neural computation
- 1969: Minsky and Papert criticize Perceptron
- 1969: Chomsky argues for universal innate grammar
- 1970: Rise of cognitive psychology and knowledge-based AI
- 1975: Learning algorithms for multi-layer neural networks
- 1985: Resurgence of neural networks and machine learning
- 1988: Birth of computational neuroscience
- 1990: Successful applications (stock market, OCR, robotics)
- 1990-2000 New synthesis of behaviorist and cognitive or representational approaches in AI and psychology
Background – Brains and Computers

- Brain consists of $10^{11}$ neurons, each of which is connected to $10^4$ neighbors.
- Each neuron is slow (1 millisecond to respond to a stimulus) but the brain is astonishingly fast at perceptual tasks (e.g., face recognition).
- Brain processes and learns from multiple sources of sensory information (visual, tactile, auditory...).
- Brain is massively parallel, shallowly serial, modular and hierarchical with recurrent and lateral connectivity within and between modules.
- If cognition is -- or at least can be modeled by -- computation, it is natural to ask how and what brains compute.

Brain and information processing

Neural Networks

Ramon Cajal, 1900
McCulloch-Pitts computational model of a neuron

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i x_i > 0 \]
\[ y = -1 \quad \text{otherwise} \]

Threshold neuron – Connection with Geometry

\[ \sum_{i=0}^{n} w_i x_i + w_0 = 0 \] describes a hyperplane which divides the instance space \( \mathbb{R}^n \) into two half-spaces.

\[ x_i \in \mathbb{R}^n : w \bullet x_i + w_0 > 0 \] and \[ x_i \in \mathbb{R}^n : w \bullet x_i + w_0 < 0 \]
McCulloch-Pitts Neuron or Threshold Neuron

\[ y = \text{sign} \left( W \cdot X + w_0 \right) \]
\[ = \text{sign} \left( \sum_{i=0}^{n} w_i x_i \right) \]
\[ = \text{sign} \left( W^T X + w_0 \right) \]

\[ \text{sign}(v) = 1 \quad \text{if} \quad v > 0 \]
\[ = 0 \quad \text{otherwise} \]

Threshold neuron– Connection with Geometry

The \((n-1)\)-dimensional hyperplane partitions the \(n\)-dimensional input space into two half spaces.

Threshold neuron – Connection with Geometry

Instance space \( \mathbb{R}^n \)

Hypothesis space is the set of \((n-1)\)-dimensional hyperplanes defined in the \(n\)-dimensional instance space.

A hypothesis is defined by \( \sum_{i=0}^{n} w_i x_i = 0 \)

Orientation of the hyperplane is governed by \( (w_1 \ldots w_n)^T \)

and the perpendicular distance of the hyperplane from the origin is given by

\[ \frac{w_0}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}} \]
Threshold neuron as a pattern classifier

The threshold neuron can be used to classify a set of instances into one of two classes $C_1, C_2$. If the output of the neuron for input pattern $X_p$ is +1 then $X_p$ is assigned to class $C_1$. If the output is -1 then the pattern $X_p$ is assigned to $C_2$.

Example:

Given $W = [w_0, w_1, w_2]^T = [-1, -1, 1]^T$,

\[ X_p^T W = [1, 0, 0] \cdot X_p + w_0 = -1 + (-1) = -2 \]

$X_p$ is assigned to class $C_2$.

Threshold neuron – Connection with Logic

Suppose the input space is $\{0, 1\}^n$.

Then threshold neuron computes a Boolean function $f: \{0, 1\}^n \rightarrow \{-1, 1\}$.

Example:

Let $w_0 = -1.5; w_1 = w_2 = 1$.

In this case, the threshold neuron implements the logical AND function.

Threshold neuron – Connection with Logic

A threshold neuron with the appropriate choice of weights can implement Boolean AND, OR, and NOT function.

Theorem: For any arbitrary Boolean function $f$, there exists a network of threshold neurons that can implement $f$.

Theorem: Any arbitrary finite state automaton can be realized using threshold neurons and delay units.

Networks of threshold neurons, given access to unbounded memory, can compute any Turing-computable function.

Corollary: Brains if given access to enough working memory, can compute any computable function.
Threshold neuron: Connection with Logic

**Theorem:** There exist functions that cannot be implemented by a single threshold neuron.

**Example:** Exclusive OR

**Why?**

---

Threshold neuron – Connection with Logic

**Definition:** A function that can be computed by a single threshold neuron is called a threshold function.

Of the 16 2-input Boolean functions, 14 are Boolean threshold functions.

As $n$ increases, the number of Boolean threshold functions becomes an increasingly small fraction of the total number of $n$-input Boolean functions:

$$N_{\text{Threshold}}(n) \leq 2^n; \quad N_{\text{Boolean}}(n) = 2^n$$

---

Terminology and Notation

**Synonyms:** Threshold function, Linearly separable function, linear discriminant function.

**Synonyms:** Threshold neuron, McCulloch-Pitts neuron, Perceptron, Threshold Logic Unit (TLU).

We often include $w_0$ as one of the components of $W$ and incorporate $x_0$ as the corresponding component of $X$ with the understanding that $x_0 = 1$. Then $y=1$ if $W.X > 0$ and $y=-1$ otherwise.
Learning Threshold functions

A training example $E_k$ is an ordered pair $(X_k, d_k)$ where

$$X_k = [x_{k1}, x_{k2}, ..., x_{kn}]$$

is an $(n+1)$ dimensional input pattern, $d_k = f(X_k) \in \{-1, 1\}$ is the desired output of the classifier and $f$ is an unknown target function to be learned.

A training set $E$ is simply a multi-set of examples.

Learning Task: Given a linearly separable training set $E$, find a solution $W^*$ such that $\forall X \in S^+$, $W^* \cdot X > 0$ and $\forall X \in S^-$, $W^* \cdot X < 0$.

Rosenblatt’s Perceptron Learning Algorithm

Initialize $W = [0, 0, ..., 0]^T$ Set learning rate $\eta > 0$

Repeat until a complete pass through $E$ results in no weight updates

For each training example $E_k \in E$

\[
\begin{align*}
    y_k & \leftarrow \text{sign} (W \cdot X_k) \\
    W & \leftarrow W + \eta (d_k - y_k) X_k
\end{align*}
\]

$W^* \leftarrow W$; Return ($W^*$)
Perceptron learning algorithm – Example

Let
\[ S^+ = \{(1, 1, 1), (1, 1, -1), (1, 0, -1)\} \]
\[ S^- = \{(1, -1, -1), (1, -1, 1), (1, 0, 1)\} \]
\[ \eta = \frac{1}{2} \]
\[ W = (0 0 0) \]

<table>
<thead>
<tr>
<th>( X_k )</th>
<th>( d_k )</th>
<th>( W )</th>
<th>( W \cdot X_k )</th>
<th>( y_k )</th>
<th>Update?</th>
<th>Updated W</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>1</td>
<td>(0, 0, 0)</td>
<td>0</td>
<td>-1</td>
<td>Yes</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>(1, 1, -1)</td>
<td>1</td>
<td>(1, 1, 1)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>(1, 0, -1)</td>
<td>1</td>
<td>(1, 1, 1)</td>
<td>0</td>
<td>-1</td>
<td>Yes</td>
<td>(2, 1, 0)</td>
</tr>
<tr>
<td>(1, -1, -1)</td>
<td>-1</td>
<td>(2, 1, 0)</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>(1, 2, 1)</td>
</tr>
<tr>
<td>(1, -1, 1)</td>
<td>-1</td>
<td>(1, 2, 1)</td>
<td>0</td>
<td>-1</td>
<td>No</td>
<td>(1, 2, 1)</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>-1</td>
<td>(1, 2, 1)</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
<td>(0, 2, 0)</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>1</td>
<td>(0, 2, 0)</td>
<td>2</td>
<td>1</td>
<td>No</td>
<td>(0, 2, 0)</td>
</tr>
</tbody>
</table>

Perceptron Convergence Theorem (Novikoff)

Theorem: Let \( \mathcal{E} = \{X_i, d_i\} \) be a training set where \( X_i \in \{0\} \times \mathbb{R}^n \) and \( d_i \in \{-1, 1\} \). Let \( S^+ = \{X_i | X_i \cdot d_i = 1\} \) and \( S^- = \{X_i | X_i \cdot d_i = -1\} \).

The perceptron algorithm is guaranteed to terminate after a bounded number \( t \) of weight updates with a weight vector \( W' \) such that \( \forall X_i \in S^+, W' \cdot X_i \geq \delta \) and \( \forall X_i \in S^-, W' \cdot X_i \leq -\delta \) for some \( \delta > 0 \), whenever such \( W' \in \mathbb{R}^{n+1} \) and \( \delta > 0 \) exist – that is, \( E \) is linearly separable. The bound on the number \( t \) of weight updates is given by
\[
  t \leq \frac{2}{\delta^2} \max_{X_i} \|X_i\| \text{ where } L = \max_{X_i} \|X_i\| \text{ and } S = S^+ \cup S^-.}
\]

Proof of Perceptron Convergence Theorem

Let \( W_t \) be the weight vector after \( t \) weight updates.

Invariant: \( \forall \theta \ |\cos \theta| \leq 1 \)
Proof of Perceptron Convergence Theorem

Let $W'$ be such that
\[ \forall x_i \in S', W' \cdot x_i \geq \delta \text{ and } \forall x_i \in S', W' \cdot x_i \leq -\delta \]
WLOG assume that $W' \cdot x = 0$ passes through the origin.

Let $\forall x_i \in S'$, $z_i = x_i,$
$\forall x_i \in S', z_i = -x_i,$
$Z = \{ z_i \}$
\( \forall x_i \in S', W' \cdot x_i \geq \delta \) \& \( \forall x_i \in S', W' \cdot x_i \leq -\delta \)
\[ \Leftrightarrow (\forall Z_i \in Z, W' \cdot z_i \geq \delta) \]
Let $E^* = \{ (Z_i, i) \}$

Proof of Perceptron Convergence Theorem

Let $w_{i+1} = w_i + \eta (y_i - y_i) Z_i$
where $w_0 = [0 \ 0 \ldots \ 0]^T$ and $\eta > 0$

[Weight update based on example $(Z_i, 1)$]
\[ \Leftrightarrow [d_i = 1 \wedge (y_i = -1)] \]
\[ \therefore W' \cdot w_{i+1} = W' \cdot (w_i + 2\eta Z_i) \]
\[ = (w' \cdot w_i) + 2\eta (W' \cdot Z_i) \]
Since $\forall Z_i \in Z, (W' \cdot Z_i \geq \delta)$ $W' \cdot w_{i+1} \geq W' \cdot w_i + 2\eta \delta$
\[ \therefore \forall t \ W' \cdot w_t \geq 2\eta \delta \]

Proof of Perceptron Convergence Theorem

\[ \|w_{i+1}\| = w_{i+1} \cdot w_{i+1}, \]
\[ = (w_i + 2\eta Z_i) \cdot (w_i + 2\eta Z_i) \]
\[ = (w_i \cdot w_i) + 4\eta (w_i \cdot Z_i) + 4\eta^2 (Z_i \cdot Z_i) \]
Note weight update based on $Z_i \Leftrightarrow (w_i \cdot Z_i \leq 0)$
\[ \therefore \|w_{i+1}\|^2 \leq \|w_i\|^2 + 4\eta^2 \|Z_i\|^2 \leq \|w_i\|^2 + 4\eta^2 L^2 \]
Hence $\|w_t\|^2 \leq 4\eta^2 L^2$
\[ \therefore \forall t \ \|w_t\| \leq 2\eta L \sqrt{t} \]

Proof of Perceptron Convergence Theorem

\[ \|w_{i+1}\|^2 = w_{i+1} \cdot w_{i+1}, \]
\[ = (w_i + 2\eta Z_i) \cdot (w_i + 2\eta Z_i) \]
\[ = (w_i \cdot w_i) + 4\eta (w_i \cdot Z_i) + 4\eta^2 (Z_i \cdot Z_i) \]
Note weight update based on $Z_i \Leftrightarrow (w_i \cdot Z_i \leq 0)$
\[ \therefore \|w_{i+1}\|^2 \leq \|w_i\|^2 + 4\eta^2 \|Z_i\|^2 \leq \|w_i\|^2 + 4\eta^2 L^2 \]
Hence $\|w_t\|^2 \leq 4\eta^2 L^2$
\[ \therefore \forall t \ \|w_t\| \leq 2\eta L \sqrt{t} \]
Proof of Perceptron Convergence Theorem

From (a) we have: \( \forall \theta \left( W \cdot W \right) \geq 2n\delta \)
\[ \Rightarrow \forall \theta \left( 2n\delta \leq \left| W \cdot W \right| \right) \Rightarrow \forall \theta \left( 2n\delta \leq \| W \| \cos \theta \right) \]
\[ \Rightarrow \forall \theta \left( 2n\delta \leq \| W \| \right) \cdot \forall \theta \left( \cos \theta \leq 1 \right) \]

Substituting for an upper bound on \( \| W \| \) from (b),
\[ \forall \theta \left( 2n\delta \leq \| W \| \right) \Rightarrow \forall \theta \left( 2n\delta \leq \| W \| \right) \]
\[ \Rightarrow \left\{ \forall t \left( t \leq \frac{\| W \|}{\delta} \right) \right\} \]

Notes on the Perceptron Convergence Theorem

- The bound on the number of weight updates does not depend on the learning rate.
- The bound is not useful in determining when to stop the algorithm because it depends on the norm of the unknown weight vector and delta.
- The convergence theorem offers no guarantees when the training data set is not linearly separable.

Exercise: Prove that the perceptron algorithm is robust with respect to fluctuations in the learning rate.

Weight space representation

Pattern space representation
- Coordinates of space correspond to attributes (features)
- A point in the space represents an instance
- Weight vector \( W \) defines a hyperplane \( W \cdot X = 0 \)

Weight space (dual) representation
- Coordinates define a weight space
- A point in the space represents a choice of weights \( W \)
- An instance \( X \) defines a hyperplane \( W \cdot X = 0 \)
Weight space representation

\[ W \cdot X = 0 \]

Solution region

\[ W \cdot X_p = 0 \quad \in S^+ \]
\[ W \cdot X_q = 0 \quad \in S^- \]

\[ W \cdot X = 0 \]

Solution region

\[ W \cdot X_p = 0 \quad \in S^+ \]
\[ W \cdot X_q = 0 \quad \in S^- \]

\[ W_{i+1} \leftarrow W_i + \eta X_p \]

Summary: Perceptron Learning Algorithm

- Simple, fast, provably convergent algorithm
- Limited to linear classifiers

Questions

- Robustness (generalization)
- Extension to non linear decision boundaries
Support Vector Machines

Improving generalization
Maximizing the margin of separation

Extension to non linear decision surfaces
Constructing separating hyperplanes in kernel-induced feature spaces

The Generalization Problem

• The curse of dimensionality: easy to overfit in high dimensional spaces
• The learning problem is ill posed (finding one hyperplane that separates the data — many such hyperplanes exist)
• Need principled way to choose the best possible hyperplane

"Capacity" of the machine — ability to learn any training set without error — related to VC dimension
Excellent memory is not an asset when it comes to learning from limited data

"A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that 'if it's green, it's a tree'"

C. Burges
Linear Learning Machines and SVM

Basic notation
- Input space \( x \in X \subseteq \mathbb{R}^n \)
- Output space \( y \in Y = \{-1, 1\} \) for classification
  \( y \in \mathbb{R} \) for regression
- Hypothesis \( h \in H \)
- Training Set \( S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \)
- Test error \( E \) also \( R(\alpha) \)
- Dot product \( \langle x, z \rangle \)

The algorithm requires that the input patterns are linearly separable, which means that there exist linear discriminant function which has zero training error. We assume that this is the case.

The Perceptron Algorithm (primal form)

initialize \( w \leftarrow 0, b \leftarrow 0, k \leftarrow 0, \eta \in \mathbb{R}^+, R \leftarrow \max \| x \| \)
repeat
  error \( \leftarrow \) false
  for \( i=1 \ldots J \)
    if \( y_i (w_i \cdot x_i) + b_i \leq 0 \)
      if \( w_i \leftarrow w_i + \eta y_i x_i \) then
        \( b_i \leftarrow b_i + \eta y_i R^2 \)
        \( k \leftarrow k + 1 \)
      end if
    end if
  end for
until (error=false)
return \( k, (w, b) \) where \( k \) is the number of mistakes
The Perceptron Algorithm Revisited

The perceptron works by adding misclassified positive or subtracting misclassified negative examples to an arbitrary weight vector, which (without loss of generality) we assumed to be the zero vector. So the final weight vector is a linear combination of training points

\[ w = \sum_{i=1}^{l} \alpha_i y_i x_i, \]

where, since the sign of the coefficient of \( x_i \) is given by label \( y_i \), the \( \alpha_i \) are positive values, proportional to the number of times, misclassification of \( x_i \) has caused the weight to be updated. It is called the embedding strength of the pattern \( x_i \).

Functional and Geometric Margin

The functional margin of a linear discriminant \((w, b)\) w.r.t. a labeled pattern \( (x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\} \) is defined as

\[ y_i \left( w \cdot x_i + b \right) \]

If the functional margin is negative, then the pattern is incorrectly classified, if it is positive then the classifier predicts the correct label.

The larger \( y_i \left( w \cdot x_i + b \right) \) the further away \( x_i \) is from the discriminant.

This is made more precise in the notion of the geometric margin which measures the Euclidean distance of a point from the decision boundary.

The geometric margin of two points

The geometric margin of a training set
Functional Margin and Geometric Margin

\[ y = \min_{i} y_i \] is called the (functional) margin of \((w, b)\) w.r.t. the data set \(S = \{(x_i, y_i)\}\).

The margin of a training set \(S\) is the maximum geometric margin over all hyperplanes. A hyperplane realizing this maximum is a maximal margin hyperplane.

Maximal Margin Hyperplane

---

Dual Representation

The decision function can be rewritten as:

\[
h(x) = \text{sgn}(\langle w, x \rangle + b) = \text{sgn}\left(\sum a_i y_i \langle x_i, x \rangle + b\right) = \text{sgn}\left(\sum a_i y_i (\langle x_i, x \rangle) + b\right)
\]

The update rule can be rewritten as:

\[
\text{if } y_j (\sum a_i y_i \langle x_i, x \rangle) + b) \leq 0 \text{ then } a_i \leftarrow a_i + \eta
\]

The learning rate \(\eta\) only influence the overall scaling of the hyperplanes, it does no effect algorithm with zero starting vector, so we can choose \(\eta = 1\).

---

Implications of Dual Representation

When Linear Learning Machines represented in the dual form

\[
f(x) = \langle w, x \rangle + b = \sum a_i y_i \langle x_i, x \rangle + b
\]

Data appear only inside dot products (in decision function and in training algorithm)

The matrix

\[
G = \left\langle \langle x_i, x \rangle \right\rangle_{i,j}
\]

is called Gram matrix

This allows for efficient use of kernel functions
### Expected Error of a Classifier

Suppose:

- We are given \( l \) observations \((x_i, y_i)\).
- Train and test points drawn randomly (i.i.d) from some unknown probability distribution \( D(x, y) \).
- The machine learns the mapping \( x \rightarrow y \) and outputs a hypothesis \( h(x, \alpha) \). A particular choice of \( \alpha \) generates “trained machine”.
- The expectation of the test error or expected risk is

\[
R(\alpha) = \frac{1}{2} \int \int |y - h(x, \alpha)| dD(x, y)
\]

### A Bound on the Generalization Performance

The empirical risk is:

\[
R_{\text{emp}}(\alpha) = \frac{1}{l} \sum_{i=1}^{l} |y_i - h(x_i, \alpha)|
\]  

Choose some \( \delta \) such that \( 0 \leq \delta \leq 1 \). With probability \( 1 - \delta \) the following bound – risk bound of \( h(x, \alpha) \) in distribution \( D \) holds (Vapnik, 1995):

\[
R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{d \log(2l/d) + 1}{2} \log(\frac{2}{\delta})}
\]

where \( d \geq 0 \) is called VC dimension is a measure of “capacity” of machine.

### A Bound on the Generalization Performance

The second term in the right-hand side is called VC confidence.

Three key points about the actual risk bound:

- It is independent of \( D(x, y) \)
- It is usually not possible to compute the left hand side.
- If we know \( d \), we can compute the right hand side.
- The risk bound gives us a way to compare learning machines!
The VC Dimension Revisited

Definition: the VC dimension of a set of functions $H = \{h(x, \alpha)\}$ is $d$ if and only if there exists a set of points $\{x_i\}_{i=1}^{d}$ such that these points can be labeled in all $2^d$ possible configurations, and for each $\{x_i\}_{i=1}^{d}$ labeling, a member of set $H$ can be found which correctly assigns those labels, but that no set exists where $q > d$ satisfying this property.

VC dimension of $H$ is size of largest subset of $X$ shattered by $H$. VC dimension measures the capacity of a set $H$ of hypotheses (functions).

If for any number $N$, it is possible to find $N$ points $x_1, \ldots, x_N$ that can be separated in all the $2^N$ possible ways, we will say that the VC-dimension of the set is infinite.

The VC Dimension Example

Suppose that the data live in $\mathbb{R}^2$ space, and the set $\{h(x, \alpha)\}$ consists of oriented straight lines, (linear discriminants). While it is possible to find three points that can be shattered by this set of functions, it is not possible to find four. Thus the VC dimension of the set of linear discriminants in $\mathbb{R}^2$ is three.
The VC Dimension of Hyperplanes

Theorem 1. Consider some set of m points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the m points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.

Corollary: The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is $n+1$, since we can always choose $n+1$ points, and then choose one of the points as origin, such that the position vectors of the remaining points are linearly independent, but can never choose $n+2$ points.

Bounds on Error of Classification

The error $\varepsilon$ of classification function $h$ for separable problem is:

$$\varepsilon = O\left(\frac{d}{L}\right)$$

Margin based bound

$$\varepsilon = O\left(\frac{1}{L}\right)$$

$$L = \max_{\mathbf{x}_i} ||\mathbf{x}_i||$$

$$\mathcal{F} = \min_{y, f(\mathbf{x})} \frac{y, f(\mathbf{x})}{||w||}$$

$$f(\mathbf{x}) = \langle w, \mathbf{x} \rangle + b$$

Maximal Margin Classifier

The bounds on error of classification suggest the possibility of improving generalization by maximizing the margin.

Minimize the risk of overfitting by choosing the maximal margin hyperplane in feature space.

SVMs control capacity by increasing the margin, not by reducing the number of features.
Maximizing Margin → Minimizing $|w|$ 

Definition of hyperplane $(w, b)$ does not change if we rescale it to $(\sigma w, \sigma b)$, for $\sigma > 0$. Functional margin depends on scaling, but geometric margin $\gamma$ does not.

If we fix (by rescaling) the functional margin to 1, the geometric margin will be equal to $\frac{1}{|w|}$.

Hence, we will maximize the margin by minimizing the norm $|w|$.

Maximizing Margin→Minimizing $|w|$ 

Distance between the two convex hulls

$$\langle w, x^+ \rangle + b = +1$$
$$\langle w, x^- \rangle + b = -1$$
$$\langle w, (x^+ - x^-) \rangle = 2$$
$$\frac{w}{||w||} (x^+ - x^-) = \frac{2}{||w||}$$
Learning as optimization

Minimize \[ \langle w, w \rangle \] (2)

subject to:

\[ y_i (\langle w, x_i \rangle + b) \geq 1 \] (3)

This is a constrained quadratic (hence convex) optimization problem with linear constraints – a quadratic programming problem.

Digression – Optimization Theory

Primal optimization problem

Given functions \( f, g_i, i=1...k; h_j, j=1...m \); defined on a domain \( \Omega \subseteq \mathbb{R}^n \);

minimize \( f(w) \) \( w \in \Omega \) \{ objective function \}

subject to:

\( g_i(w) \leq 0 \) \( i = 1...k \) \{ inequality constraints \}

\( h_j(w) = 0 \) \( j = 1...m \) \{ equality constraints \}

Shorthand:

\( g(w) \leq 0 \) denotes \( g_i(w) \leq 0 \) \( i = 1...k \)

\( h(w) = 0 \) denotes \( h_j(w) = 0 \) \( j = 1...m \)

Feasible region \( F = \{ w \in \Omega : g(w) \leq 0, h(w) = 0 \} \)

Maximum Margin Hyperplane

The problem of finding the minimal margin hyperplane is a constrained quadratic optimization problem which can be solved using theory developed by Lagrange (and extended by Karush, Kuhn, and Tucker).

Lagrangian:

\[ L_\alpha(w) = \frac{1}{2} \langle w, w \rangle - \sum \alpha_i [y_i (\langle w, x_i \rangle + b) - 1] \] (4)

\( \alpha \geq 0 \)
SVM Solution

The optimal solutions must satisfy

\[ \alpha \cdot (w, b) \]

\[ \alpha_i [y_i (\langle w, x_i \rangle + b) - 1] = 0 \quad \forall i \tag{16} \]

Only the training samples \( x_i \) for which the functional margin = 1 have the nonzero \( \alpha_i \). They are called Support Vectors.

The optimal hyperplane can be expressed in the dual representation in terms of this subset of training samples – the support vectors

\[ f(x, \alpha, b) = \sum_{i=1}^{l} y_i \alpha_i \langle x, x_i \rangle + b = \sum y_i \alpha_i \langle x, x_i \rangle + b \]

Support Vector Machines Yield Sparse Solutions

![Diagram](image.png)
Extending Linear Classifiers – Kernel Machines

Map data into a feature space where they are linearly separable \( x \to \phi(x) \)

Learning in the Feature Space

Consider a target function describing the gravitational force between two bodies. 

\[ f(m_1, m_2, r) = \frac{G m_1 m_2}{r} \]

Observable quantities are masses \( m_1, m_2 \) and distance \( r \). A linear machine could not represent it, but a change of coordinates 

\[ (m_1, m_2, r) \to (x_1, x_2, r) = (\ln m_1, \ln m_2, \ln r) \]

gives the representation

\[ g(x_1, x_2, r) = \ln f(m_1, m_2, r) = \ln G + \ln m_1 + \ln m_2 - 2\ln r = c + x_1 + x_2 - 2z \]

Learning in the Feature Spaces

High dimensional feature spaces

\[ x = (x_1, \ldots, x_n) \to \phi(x) = (\phi_1(x), \ldots, \phi_d(x)), d < n \]

solve the problem of expressing complex functions

But this introduces the

- computational problem (working with very large vectors)
- generalization theory problem (curse of dimensionality)
**Implicit Mapping to Feature Space**

Kernel Machines
- Solve the computational problem of working with many dimensions
- Can make it possible to use infinite dimensions efficiently
- Offer other advantages, both practical and conceptual

**Kernel-Induced Feature Spaces**

\[ f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b \]

where \( \phi: X \rightarrow F \) is a non-linear map from input space to feature space.

In the dual representation, the data points only appear inside dot products.

**Kernels**

Kernel function returns the value of the dot product between the images of the two arguments

\[ K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \]

When using kernels, the dimensionality of space \( F \) not necessarily important. We may not even know the map \( \phi \).

Given a function \( K \), it is possible to verify that it is a kernel.
Kernel Machines

We can use perceptron learning algorithm in the feature space by taking its dual representation and replacing dot products with kernels:

$$\langle x_1, x_2 \rangle \leftarrow K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

The Kernel Matrix (Gram Matrix)

$$K = \begin{bmatrix}
K(1,1) & K(1,2) & K(1,3) & \ldots & K(1,m) \\
K(2,1) & K(2,2) & K(2,3) & \ldots & K(2,m) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
K(m,1) & K(m,2) & K(m,3) & \ldots & K(m,m)
\end{bmatrix}$$

Properties of Kernel Functions - Mercer’s Theorem

- The kernel matrix is Symmetric Positive Definite
- Any symmetric positive definite matrix can be regarded as a kernel matrix, that is, as an inner product matrix in some space
- Mercer’s Theorem: Every (semi) positive definite, symmetric function is a kernel: i.e. there exists a mapping $\phi$ such that it is possible to write:

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$
Properties of Kernel Functions - Mercer’s Theorem

Eigenvalues of the Gram Matrix define an expansion
expansion Mercer’s Kernels:

\[ K(x_i, x_j) = \sum \lambda_i \phi_i(x_i), \phi_i(x_j) \]

That is, the eigenvalues act as features!

Examples of Kernels

Simple examples of kernels:

\[ K(x, z) = (x, z)^d \]
\[ K(x, z) = e^{-\|x-z\|^2 / 2\sigma} \]

Example: Polynomial Kernels

\[ x = (x_1, x_2)_k, \]
\[ z = (z_1, z_2)_k, \]
\[ \langle x, z \rangle = (x_1z_1 + x_2z_2)^k \]
\[ = x_1^k z_1^k + x_2^k z_2^k + 2x_1z_1x_2z_2 \]
\[ = \langle \phi(x), \phi(z) \rangle \]
Example: Polynomial Kernels

$$\phi(x)$$

Making Kernels

The set of kernels is closed under some operations. If \( K, K' \) are kernels, then:
- \( K + K' \) is a kernel
- \( cK \) is a kernel, if \( c > 0 \)
- \( aK + bK' \) is a kernel, for \( a, b > 0 \)

One can make complex kernels from simple ones: modularity!

Kernel Machines and SVM

Kernel Machines are Learning Machines, that:
- Use a dual representation
- Operate in a kernel induced feature space (that is a linear function in the feature space implicitly defined by \( K \))

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$$

SVM are kernel machines that find a maximum margin separating hyperplane in a kernel induced high dimensional feature space
SVM Implementation

- Use QP packages (MINOS, LOQO, quadprog from MATLAB optimization toolbox). - Require that the data are held in memory in the form of kernel matrix
- Specialized algorithms for SVM – including gradient based methods that do not require all of the data to be available in memory

On-line algorithm for the 1-norm soft margin

Given training set D and learning rates $\eta \in \mathbb{R}^l$

$\alpha \leftarrow 0$

repeat

for $i = 1$ to $l$

$\alpha_i \leftarrow \alpha_i + \eta (1 - y_i \sum \alpha_j y_j K(x_i, x_j))$

if $\alpha_i < 0$ then $\alpha_i \leftarrow 0$

else if $\alpha_i > C$ then $\alpha_i \leftarrow C$

end for

until stopping criterion satisfied

return $\alpha$

Applications of SVM

- Face recognition
- Character recognition
- Text classification
- Protein Function Classification
- Protein-Protein interface identification
Prediction of Protein-Protein Interfaces

Can we use machine learning approaches to discover sequence correlates predictive of protein-protein interaction sites?

Changhui Yan, Feihong Wu, Drena Dobbs, Robert Jernigan, Vasant Honavar,

Protein-Protein Interactions

- Protein function depends on interactions with other proteins and ligands
- Determinants of protein-protein interaction sites include residue hydrophobicity, charge, solvent accessibility, etc.
- Interfaces in different types of complexes (e.g., homodimers vs heterodimers) have different properties
- There are several structure-based approaches for predicting protein-protein interaction sites
- Sequence-based approaches are needed!
- Challenge: protein interaction sites often involve long-range interactions, i.e., contacts between groups of amino acids that are widely separated in primary amino acid sequence
Data-Driven Protein Interaction Prediction

- Given that a protein interacts with another protein, can we predict which residues are in the interface?
- Can we build a reliable predictor by training on datasets of known heterocomplexes?

Sequence-based Prediction of Interface Residues

Given only the sequence of a target protein, can we predict which residues form the protein-protein interface?

**Goal:** classify amino acid residues into *interface* and *non-interface* residues

Machine Learning Approach

- Divide empirical data into disjoint training and test sets
- Train classifier (SVM, Naive Bayes, etc.) to discover relationship between local sequence features and functionality of target residues
- Test classifier on test set
- Apply classifier to novel data
Datasets

Non-redundant sets of heterocomplexes from PDB [Chakrabarti et al. 2002]

Total of 70 heterocomplexes (77 distinct proteins) from six categories:
- antibody-antigen (AA)
- protease-inhibitor (PI)
- enzyme-inhibitor
- G-protein, other signal transduction
- large protease
- miscellaneous

Extract Interface Residues from Complexes

Interface residues = amino acids with reduction of at least 1 Å2 in solvent accessible surface area (ASA) in complex relative to monomer

Positive examples: Interface residues
Negative examples: Non-interface residues

Support Vector Machine (SVM)

Support vector machine implements a linear discriminant which separates input patterns into two classes.

If the classes are not linearly separable, nonlinear kernel functions can be used to implicitly map patterns into a higher (possibly even infinite) dimensional space in which they are separable.

Among all separating hyperplanes, SVM finds one that maximizes the margin of separation.
Data Representation

Input: An 9-residue contiguous window with target residue at center.

Performance of the SVM classifier

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr Coeff</td>
<td>0</td>
<td>-0.02</td>
<td>0.19</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.53</td>
<td>0.51</td>
<td>0.66</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.31</td>
<td>0.30</td>
<td>0.44</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.37</td>
<td>0.42</td>
<td>0.43</td>
</tr>
</tbody>
</table>

Sequence features are predictive of interaction

Accuracy of SVM classifier trained on original data set significantly higher than that of classifier trained on randomly labeled examples

- Local sequence features are predictive target residue function
- SVM (and other machine learning algorithms) can discover sequence features that are reliable predictors of interface residues

But.. Can we do better?
Key Observation: Interface residues are usually located in small clusters within primary amino acid sequence.

The Schematic of the 2-stage classifier:

- Interface residue
- SVM
- Bayes
- The second stage exploits the fact that interface residues tend to be clustered along the sequence
- Position relative to an interface residue

2-stage classifier outperforms 1-stage SVM:

<table>
<thead>
<tr>
<th></th>
<th>SVM method</th>
<th>Two-stage method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.19</td>
<td>0.30</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.66</td>
<td>0.72</td>
</tr>
<tr>
<td>Specificity $^*$</td>
<td>0.44</td>
<td>0.58</td>
</tr>
<tr>
<td>Sensitivity $^*$</td>
<td>0.43</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Yan, Dobbs, and Honavar, 2004 (ISMB, Bioinformatics)
Summary

- Local sequence features are predictive of protein interaction sites
- SVM and Naive Bayes can successfully discover and use such features to identify interface residues

Prologue – More recently, we have developed a probabilistic graphical model that achieves a correlation coefficient of 0.66 and accuracy of 0.89
Lazy Learners

- Are conceptually simple
- Asymptotically have error rates that are no worse than twice that of the optimum Bayes classifier
- Learn by simply memorizing training examples
- Construct a different classifier for each input instance (query instance) unlike eager learning algorithms which construct a single classifier during the learning phase and use it for classifying each query instance.

Lazy Learning or Instance Based Learning

- The computational effort of learning is low
- The storage requirements of learning are high – need to memorize the examples in the training set
- Cost of classifying new instances can be high
- A distance measure needs to be defined over the input space – e.g., Euclidean distance, Hamming distance, etc. as appropriate
- Performance degrades when there are many irrelevant attributes
K nearest neighbor method

Instances are assumed to lie in an \( n \)-dimensional instance space – e.g., the Euclidean space.

An instance \( X \) is described by a feature vector

\[ X_p = [x_{i_1}, \ldots, x_{i_n}] \]

Where \( x_{i_p} \) denotes the value of the \( i_p \)th feature in \( X_p \).

\[ d(X', X) = \left( \sum_{i=1}^{n} (x_i - x'_i)^2 \right)^{1/2} \]

Defines the Euclidean distance between two patterns in the Euclidean space – other distance measures can be used as needed.

K nearest neighbor Classifier

Learning Phase

For each training example \((X_i, f(X_i))\), store the example in memory.

Classification phase

Given a query instance \( X_q \), identify the \( k \) nearest neighbors \( X_1, \ldots, X_k \) of \( X_q \).

\[ g(X_q) = \arg \max \sum_{i=1}^{k} \delta(a, f(X_i)) \]

where \[ \delta(a, b) = 1 \text{ iff } a = b \text{ and } \delta(a, b) = 1. \]

K nearest neighbor Function Approximator

Learning Phase

For each training example \((X_i, f(X_i))\), store the example in memory.

Approximation phase

Given a query instance \( X_q \), identify the \( k \) nearest neighbors \( X_1, \ldots, X_k \) of \( X_q \).

\[ g(X_q) = \sum_{i=1}^{k} \frac{f(X_i)}{k} \]
Distance weighted K nearest neighbor Classifier

Learning Phase
For each training example \((X_i, f(X_i))\), store the example in memory

Classification phase
Given a query instance \(X_q\), identify the \(k\) nearest neighbors of \(X_q\) - KNN

\[ g(X_q) = \text{argmax} \sum_{i=1}^{k} w(X_q) \delta(o, f(X_i)) \]
where
\[ w_i = \frac{1}{d(X_q, X_i)} \]

Decision Boundary induced by the 1 nearest neighbor classifier

Voronoi Diagram
Locally weighted regression

Locally weighted regression involves calculating an approximation of the function value for a given input based on its nearest neighbors when needed during the approximation phase as opposed to during the learning phase.

Let the approximation be of the form

\[ g(X) = w_0 + \sum_{i=1}^{N} w_i x_i \]

in a small neighborhood around a query \( X_q \)

Minimize the error over the K nearest neighbors of \( X_q \)

\[
E(X_q) = \frac{1}{2} \sum_{x \in \text{NN}(X_q)} (f(x) - g(x))^2
\]

\[
w_i \leftarrow w_i - \eta \frac{\partial E(X_q)}{\partial w_i}
\]

\[
w_i \leftarrow w_i + \eta \sum_{x \in \text{NN}(X_q)} (f(x) - g(x)) x_i
\]

Minimize the error over all the neighbors of \( X_q \) in the training set weighted by an inverse function of distance to the neighbors

\[
E_i(X_q) = \frac{1}{2} \sum_{x \in \text{NN}(X_q) \cap D} (f(x) - g(x))^2 \phi(d(X_q, x))
\]

\[
w_i \leftarrow w_i - \eta \frac{\partial E_i(X_q)}{\partial w_i}
\]

\[
w_i \leftarrow w_i + \eta \sum_{x \in \text{NN}(X_q) \cap D} \phi(d(X_q, x)) (f(x) - g(x)) x_i
\]
Locally weighted regression

Minimize the error over the $K$ nearest neighbors of $X_q$ in the training set weighted by an inverse function of distance to the neighbors:

$$E_q(X_q) = \frac{1}{2} \sum_{x \in \text{NN}(X_q)} (f(x) - g(x))^2 \phi(d(x, X_q))$$

$$w_i \leftarrow w_i - \frac{\partial E_q(X_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{x \in \text{NN}(X_q)} \phi(d(x, X_q))(f(x) - g(x)) s.$$
Summary

Machine learning algorithms provide some of the most powerful tools for discovering knowledge and generating experimentally testable hypotheses from macromolecular sequence, structure, expression, and function data.

Since a large part of bioinformatics and computational biology has to do is to turn large amounts of data into knowledge, hypotheses, and theories, some familiarity with machine learning algorithms is essential to every computational biologist.

What we have seen here is only the tip of the iceberg.

Summary

What we have seen here is only the tip of the iceberg.

No single algorithm works best for every application.

Some simple algorithms are effective on many data sets.

Better results can be obtained by preprocessing the data to suit the algorithm or adapting the algorithm to suit the characteristics of the data.

Some current research challenges include learning from large, heterogeneous, distributed, autonomous data and knowledge sources.

Further Information


Course on Machine Learning – http://www.cs.iastate.edu/~cs573x/

Textbooks


Artificial Intelligence Research Laboratory – http://www.cs.iastate.edu/~honavar/aigroup.html
### Further Information

#### Conferences
- International Conference on Machine Learning (ICML)
- International Conference on Data Mining (ICDM)
- ACM Conference on Knowledge Discovery and Data Mining (KDD)
- Intelligent Systems in Molecular Biology (ISMB)
- International Joint Conference on Artificial Intelligence (IJCAI)
- National Conference on Artificial Intelligence (AAAI)
- Intelligent Data Analysis (IDA)
- European Conference on Machine Learning (ECML)
- Pacific Conference on Data Mining and Knowledge Discovery (PKDD)

#### Journals
- Bioinformatics Journal
- Machine Learning Journal
- Artificial Intelligence Journal
- Journal of Machine Learning Research
- Journal of Artificial Intelligence Research
- IEEE Transactions on Data and Knowledge Engineering
- IEEE Transactions on Neural Networks