Data Mining

- Data mining emerged in the 1980's when the amount of data generated and stored became overwhelming.
- Data mining is strongly influenced by other disciplines such as mathematics, statistics, artificial intelligence, data visualization, etc.
- One of the difficulties with it being a new area is that the terminology is not fixed; the same concept may have different names when used in different applications.
- We first see how Data Mining compares with other areas.
- Remember that we are using the working definition:

  "Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data." (W. Frawley).

Data Mining vs Statistics

Statistics can be viewed as a mathematical science and practice of developing knowledge through the use of empirical data expressed in quantitative form. Statistics allows us to discuss randomness and uncertainty via probability theory. For example, statisticians might determine the covariance of two variables to see if these variables vary together and measure the strength of the relationship. But data mining strives to characterize this dependency on a conceptual level and produce a causal explanation and a qualitative description of the data. Although data mining uses ideas from statistics it is definitely a different area.

Data Mining vs Machine Learning

Machine Learning is a subfield of artificial intelligence which emerged in the 1960's with the objective to design and develop algorithms and techniques that implement various types of learning. It has applications in areas as diverse as robot locomotion, medical diagnosis, computer vision, handwriting recognition, etc. The basic idea is to develop a learning system for a concept based on a set of examples provided by the "teacher" and any background knowledge. Main types

are supervised and unsupervised learning (and modifications of these). Machine Learning has influenced data mining but the areas are quite different. Dr. Barbu in the Statistics Department offers a course in Machine Learning.

Data Mining vs Knowledge Discovery from Databases (KDD)

The concept of KDD emerged in the late 1980's and it refers to the broad process of finding knowledge in data. Early on, KDD and Data Mining were used interchangeably but now Data Mining is probably viewed in a broader sense than KDD.

Data Mining vs Predictive Analytics

Wikipedia's definition is " predictive analytics encompasses a variety of techniques from statistics, data mining and game theory that analyze current and historical facts to make predictions about future events." The core of predictive analytics relies on capturing relationships between explanatory variables and the predicted variables from past occurrences and exploiting it to predict future outcomes. One aspect of Data Mining is predictive analytics.

Stages of Data Mining

1. Data gathering, e.g., data warehousing, web crawling
2. Data cleansing - eliminate errors and/or bogus data, e.g., patient fever = 125
3. Feature extraction - obtaining only the interesting attributes of the data, e.g., date acquired is probably not useful for clustering celestial objects
4. Pattern extraction and discovery - this is the stage that is often thought of as data mining
5. Visualization of the data
6. Evaluation of results; not every discovered fact is useful, or even true! Judgement is necessary before following your software's conclusions.

Clearly we can't look at all aspects of Data Mining but we'll just pick a few and get the basic idea.

Dr. Meyer-Baese gives a course in Data Mining if you are interested in learning more about the topic.

- Clustering for feature extraction - we have already talked about this
- Classification - algorithms to assign objects to one of several predefined categories
- Association Rules - algorithms to find interesting associations among large sets of data items.
- Neural Networks
- Support Vector Machine
- Genetic Algorithms

Classification

Examples include:

- classifying email as spam based upon the message header and content
- classifying cells as benign or cancerous based upon results of scan
- classifying galaxies as e.g., spiral, elliptical, etc. based on their shapes
- classifying consumers as potential customers based upon their previous buying

Terminology

Each record is known as an instance and is characterized by the attribute set, $\mathbf{x}$ and a target attribute or class label y

When we use Classification we attempt to find a target function f that maps each attribute set $\mathbf{x}$ to one of the predefined class labels y. The target function is also called the classification model.

Typically we will use a set of data, called the training set, to build our model.

We can use the target function for one of two purposes:

- Descriptive Modeling - Goal is to serve as an explanatory tool to distinguish between objects of different classes.
- Predictive Modeling - Goal is to predict the class label for unknown records.

There are 4 types of attributes:

- nominal - different names; e.g., brown or blue for eye color, SSN, gender

- ordinal - provides ordering; e.g., hot, mild, cold; small, medium, large
- interval - difference in values are meaningful; e.g., dates, temperature
- ratio- differences and ratios are meaningful; e.g., mass, age

Attributes can be discrete or continuous.

Discrete attributes can be categorical such as zip codes, SSN or just numerical. Binary attributes are a special case of discrete and only take on two values such as married or not, homeowner or not, mammal or not, etc. These can be represented as 0 and 1 and are often called Boolean attributes.

Continuous attributes have values which are real numbers; e.g., temperature, weight, salary, etc.

Classification techniques are best suited for data which is binary or nominal. Often when we have continuous data we transform it to ordinal such as small, medium, or large.

General Approach to Solving a Classification Problem

The goal is to use a systematic approach to build a classification model from our training data. The model should fit the training data well and correctly predict the class labels of unseen records not in the training data.

We may use

- decision tree classifiers
- rule-based classifiers
- neural networks
- support vector machine
- Bayes classifiers
- ...

Each technique uses a learning algorithm to identify a model that best fits the

relationship (in some sense) between the attribute set and class label for the input data.

General Steps

1. Provide a training set of records whose class labels are known
2. Apply one of the techniques above to build a classification model using the training set
3. Apply the model to a test set to determine class labels
4. Evaluate the performance of the model based on the number of correct/incorrect predictions of the test set; we can then determine the accuracy as the fraction of correct predictions or the error rate as the fraction of wrong predictions.

Example Suppose we want to classify records as either Class A or Class B. We use our classification model on our test set and get the following results.

Actual Class	Predicted Class	
	Class A	Class B
Class A	43	10
Class B	12	35

In this test set there are 100 records. The table says that 43 records were correctly labeled as Class A and 10 records were incorrectly labeled as Class A. Also 35 Class B records were correctly labeled and 12 were mislabeled as Class A. This means that our accuracy is 78/100 or 78% and our error is 22/100 or 22%.

So now what we need to do is find a way to build a classification model. We will look at decision trees which is probably the easiest approach.

Decision Trees

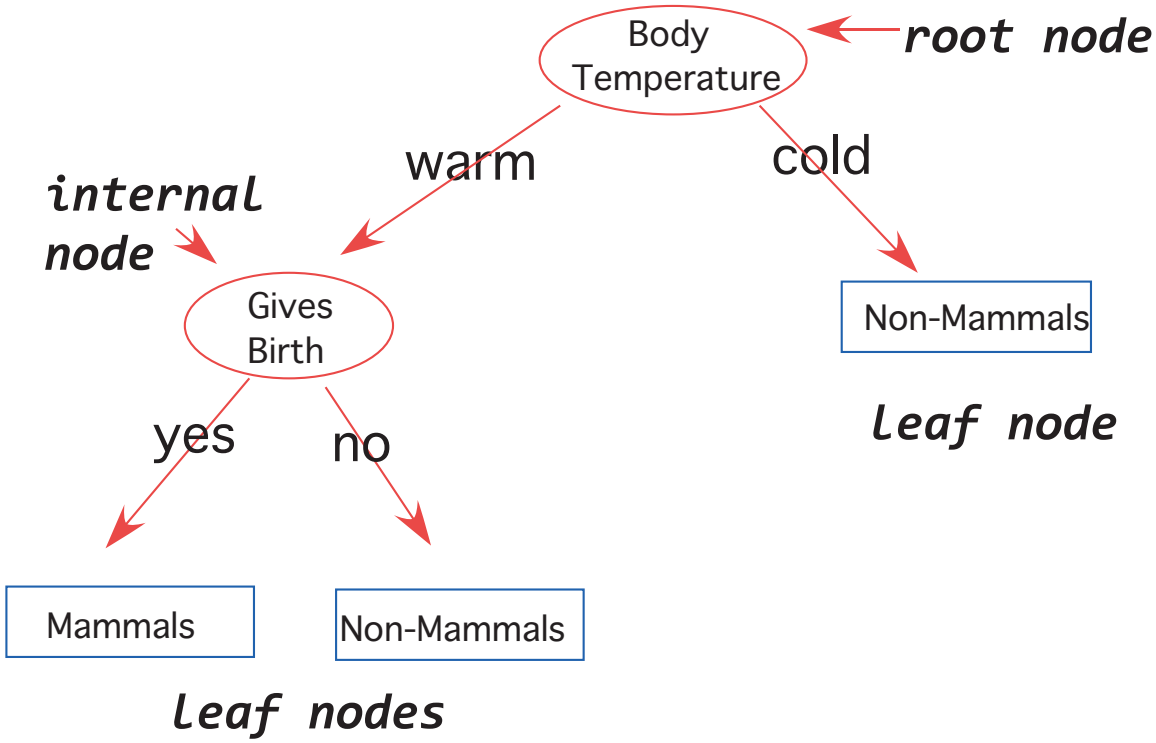
The idea behind decision trees is to pose a series of questions about the characteristics we want. Of course we must carefully choose the questions in order to develop the desired attributes.

Example Suppose we have a list of vertebrates and we want to classify them as mammals or non-mammals. Below is a possible decision tree for classifying a vertebrate. Note the following terminology:

root node - no incoming edges and zero or more outgoing edges

internal node - exactly one incoming edge and two or more outgoing edges

leaf node - exactly one incoming and no outgoing



Suppose we want to use the decision tree to classify a penguin which has the following attributes:

name	body temp	gives birth	class
penguin	warm-blooded	no	?

Applying the decision tree we see that the penguin is classified as a non-mammal because it is warm-blooded but doesn't give birth.

If we think about it we realize that there are exponentially many decision trees that can be constructed from a given set of attributes. So what do we do? Finding the optimal one is probably not an option so we settle for a suboptimal result.

Many decision trees are inductive and use a greedy approach.

A greedy algorithm is one which constructs a solution through a sequence of steps where at each step the choice is made based upon the criteria that

(i) it is the best local choice among all feasible choices available at that step and

(ii) the choice is irrevocable, i.e., it cannot be changed on subsequent steps of the algorithm.

Example Suppose we want to build a decision tree to predict whether a person will default on his/her car loan payments. We collect data from previous bor-

rowers and accumulate the following training set. The attributes we summarize are: (i) homeowner (binary attribute), (ii) marital status (nominal/categorical), (iii) annual income (continuous). Our target class label is binary and is whether that person defaults on the loan payments.

#	home owner	marital status	annual income	defaulted
1	yes	single	125K	no
2	no	married	100K	no
3	no	single	70K	no
4	yes	married	120K	no
5	no	divorced	95K	yes
6	no	married	60K	no
7	yes	divorced	220K	no
8	no	single	85K	yes
9	no	married	75K	no
10	no	single	90K	yes

Hunt's algorithm grows a decision tree in a recursive manner. The records are subsequently divided into smaller subsets until all the records belong to the same

class.

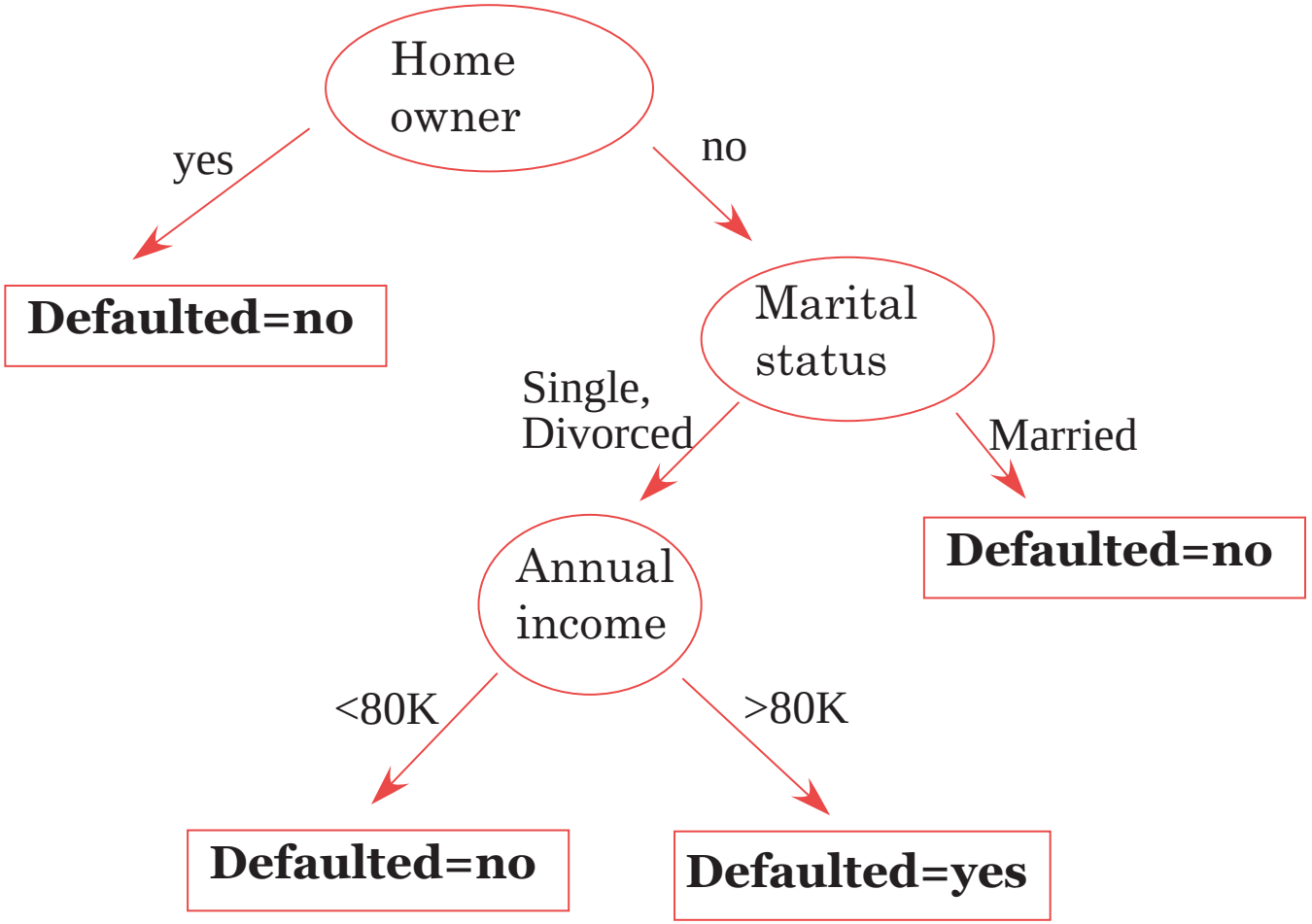
Step 0 - check to make sure all records in training set didn't answer "no" or all answer "yes" to "defaulted". In our training set there were individuals who defaulted and those that didn't.

Step 1 - determine your first criteria for making the "greedy" choice. Here we choose the attributes in order and choose home ownership. We note that all three home owners did not default on their loans so that is a leaf; however some non-home owners defaulted and others didn't so we need to subdivide further.

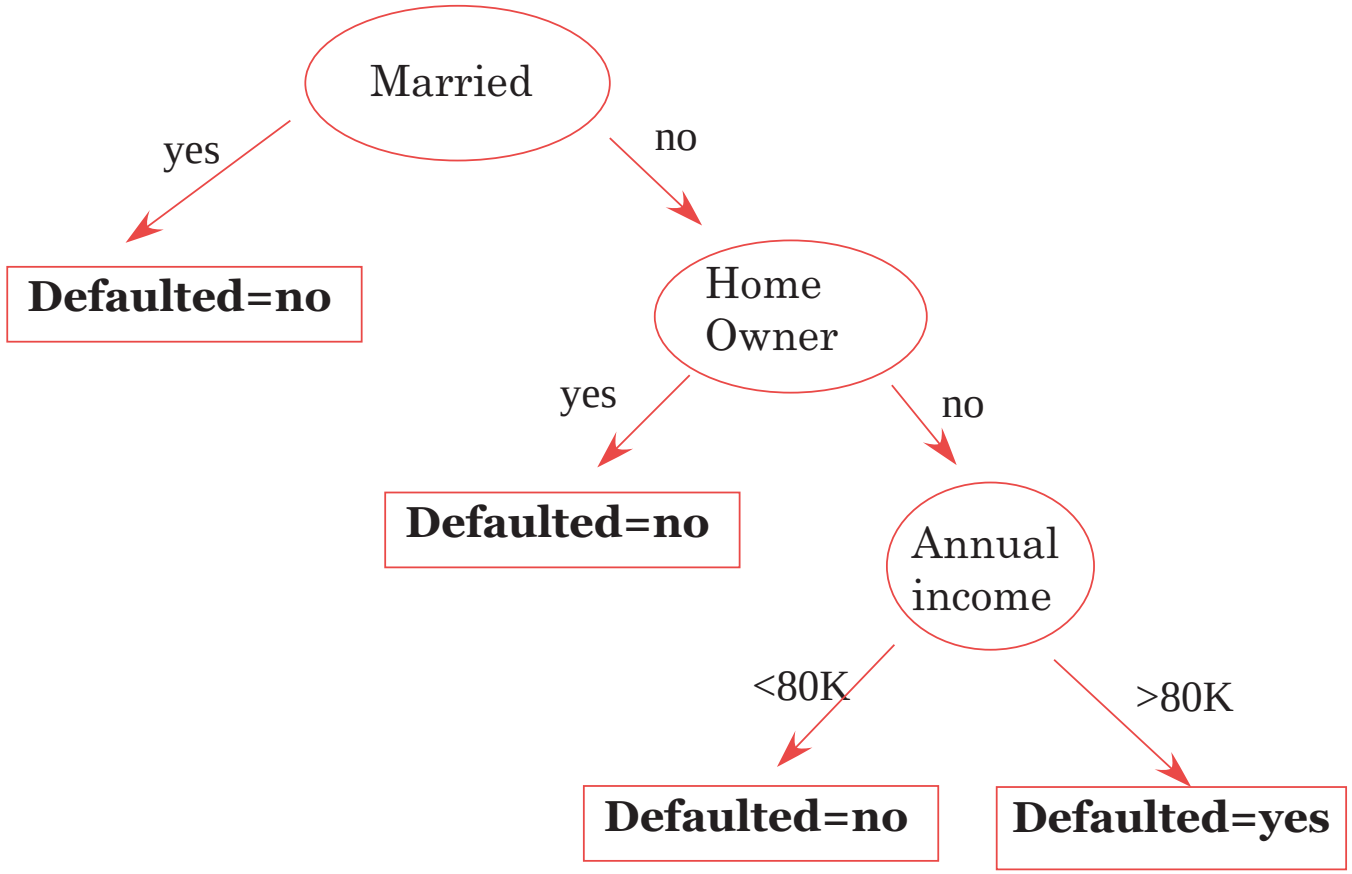
Step 2 - our second criteria is marital status. Here we note that all married borrowers repaid their loans so that is a leaf; however all single and divorced did not repay so we need to subdivide again.

Step 3 - our third criteria is annual income. The group of non-homeowners who are single or divorced is divided by $< 80K$ or $> 80K$. In this case the individuals making more than 80K defaulted and those making less did not.

The resulting decision tree is illustrated below.



Of course if we ask the questions in a different order we get a different decision tree as the following demonstrates.


Married
yes
no
Defaulted=no
Home Owner
yes
no
Defaulted=no
Annual income
<80K
>80K
Defaulted=no
Defaulted=yes

Hunt's algorithm for determining a decision tree

We have seen that the approach is recursive and at each step we partition the training records into successively similar subsets.

To describe the method we let $y = \{y_1, y_2, \ldots, y_\ell\}$ be the class labels (in our case we just have default yes and default no). Let $\mathcal{D}_i$ be the ith subset of the training set that is associated with node i (either root, internal or leaf node).

The algorithm is applied recursively as follows:

Check to see if all records in $\mathcal{D}_i$ below to the same class y_i.

- If so, then i is a leaf node (i.e., terminal)
- If not, then choose an attribute test condition to partition the records into smaller subsets. A child node (internal node) is created for each outcome of the test condition and the records in $\mathcal{D}_i$ are distributed according to the outcome of the test condition.

Of course one of these child nodes may be empty if none of the training records have the combination of attributes associated with each node. In this case we just declare it a leaf node with the same class label as the majority class of training records associated with its parent node.

Also suppose we had separated our home owners and the ones who owned homes had identical attributes but different class labels, i.e., some defaulted and some didn't. We couldn't separate these records any further. In this case we declare it a leaf node with the same class label as the majority.

How should we stop the tree growth?

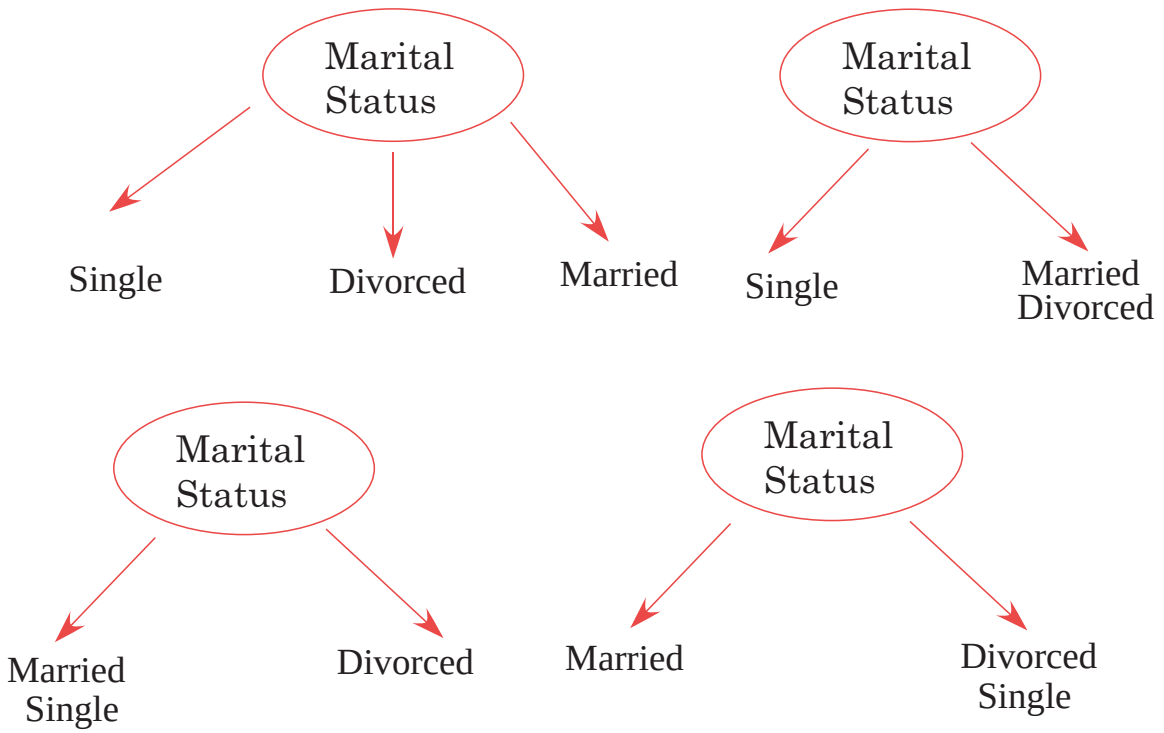
We need a termination criteria for our recursive algorithm. We could stop it when either all records in $\mathcal{D}_j$ have the same class label or are empty or all have identical attributes except for the class label. However, there may be times when it is advantageous to terminate early.

How should we split each training set?

At each recursive step we must select an attribute test condition to divide the records into smaller subsets. So we want to investigate how to choose a test condition to divide $\mathcal{D}_i$. Can we make a more intelligent choice than a random selection? Let's look at the situation for different types of attributes.

Binary attributes are in a sense the easiest because they only generate two potential outcomes; e.g., a home owner query is either yes or no.

Nominal attributes can have many values so splitting them can result in more than two child nodes or we can split it by grouping all but one value in one child node. For example, if we query marital status we can have the following splits.



Ordinal attributes can produce two or more splits; e.g., small, medium, large.

Continuous attributes are usually tested with a comparison, i.e., $\leq$, $>$

So now suppose we are at a step of our algorithm and want to determine which attribute to use as a test. What we would like is a measure for selecting the best

way to divide our records.

Let's look at the easiest case of binary attributes with only two classes (like mammal or non-mammal or default or no default).

Let $p(i|t)$ denote the fraction of records belonging to class i at a given node t; so in the two class problem $p(1) + p(2) = 1$.

When we split $\mathcal{D}_t$ then we would like at least one of the child nodes to be "pure" or homogeneous in the sense that all records in that node are of the same class. So it is reasonable to use a measure of the "impurity" or heterogeneity of the child nodes which we split $\mathcal{D}_t$.

To this end, the following measures are often used for a node t; here k is the number of classes.

$$\text{Gini}(t) = 1 - \sum_{i=1}^{k} \big(p(i|t)\big)^2$$

$$\text{Classification error}(t) = 1 - \max_{1 \le i \le k} \big(p(i|t)\big)$$

$$\text{Entropy}(t) = -\sum_{i=1}^{k} \big(p(i|t)\big) \log_2 p(i|t)$$

The first two are related to standard norms. To understand the entropy measure consider the case of two variables like a coin toss. The outcome of a series of coin tosses is a variable which can be characterized by its probability of coming up heads. If the probability is 0.0 (tails every time) or 1.0 (always heads), then there isn't any mix of values at all. The maximum mix will occur when the probability of heads is 0.5 which is the case in a fair coin toss. Let's assume that our measure of mixture varies on a scale from 0.0 ("no mix") to 1.0 ("maximum mix"). This

means that our measurement function would yield a 0.0 at a probability of 0.0 (pure tails), rise to 1.0 at a probability of 0.5 (maximum impurity), and fall back to 0.0 at a probability of 1.0 (pure heads). This is what the Entropy measures does.

Example

Suppose we have 20 records (10 male and 10 female) and our classes are "shop at Overstock.com" or not (say class 1 and class 2) and we have divided the records by gender. For different scenarios of the female "child" node we want to compute the three measurements of error.

(i) all 10 females are of class 1

Because $p(1) = 1.0$ and $p(2) = 0.0$ we have

$$\text{Gini}(t) = 1 - (1^2) = 0$$

$$\text{Classification error}(t) = 1 - 1 = 0.0$$

$$\text{Entropy}(t) = -(1 \log_2(1) + 0) = 0.0$$

and as expected, the "impurity" measures are zero, i.e., the results are homogeneous.

(ii) 5 females are of class 1 and 5 of class 2

Because $p(1) = 0.5$ and $p(2) = 0.5$ we have

$$\text{Gini}(t) = 1 - (.5^2 + .5^2) = 0.5$$

$$\text{Classification error}(t) = 1 - 0.5 = 0.5$$

$$\text{Entropy}(t) = -\big(.5 \log_2(.5) + .5 \log_2(.5)\big) = 1.0$$

These are the maximum values that the measures take on because the class is equally split so it is the least homogeneous.

(ii) 8 females are of class 1 and 2 of class 2

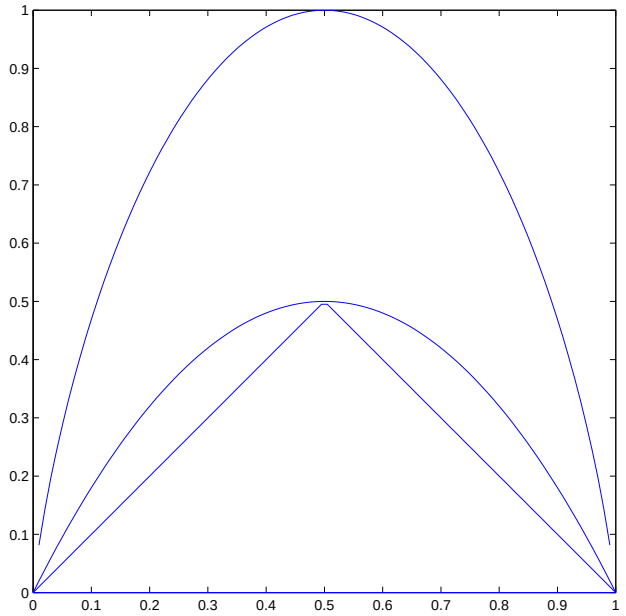
Because $p(1) = 0.8$ and $p(2) = 0.2$ we have

$$\text{Gini}(t) = 1 - (.8^2 + .2^2) = 0.32$$

$$\text{Classification error}(t) = 1 - 0.8 = 0.2$$

$$\text{Entropy}(t) = -\big(.8\log_2(.8) + .2\log_2(.2)\big) = 0.7219$$

If we were to plot these quantities for a range of probabilities we would get that they achieve their maximum when there is a uniform class distribution. This is shown in the figure below.



To determine how a test condition performs, we compare the degree of impurity of the parent node before splitting with the degree of impurity of the child nodes after splitting. The larger their difference the better the test condition.

The gain $\triangle$ is a measure that can be used to determine how good a split we are making so our goal will be to choose a test criterion that will maximize the gain. Of course it will depend on the measure we use. Basically all we do is take the difference in the impurity measure of the parent minus a weighted average of the measures of each child node.

Gain: Let I denote the impurity measure (i.e., one of the measurements defined above). Assume that we have split the parent node which consists of N records into k child nodes which consist of N_j records in the jth child node. Then

$$\Delta = I_{\text{parent}} - \sum_{j=1}^{k} \frac{N_j}{N} I_j$$

When the entropy measurement is used it is known as the information gain.

Example Suppose we have a parent node which is equally split so $I_{\text{parent}} = 0.5$ for Gini measure. Now let's say we use two different criteria to split the records and we get two child nodes with the following results.

Criteria A

Node 1 - 4 in class 1 and 3 in class 2 Node 2 - 2 in class 1 and 3 in class 2

Criteria B

Node 1 - 1 in class 1 and 4 in class 2 Node 2 - 5 in class 1 and 2 in class 2

Which criterion is better? Let's use the Gini measure and compare.

We compute the Gini measure for Node 1 to get 0.4898 and for Node 2 we get 0.480 so the gain for using attribute A as a query is the weighted average

$$.5 - \left(\frac{7}{12}(.4898) + \frac{5}{12}(.480)\right) = .5 - 0.4857 = 0.0143$$

For criteria B we compute the Gini measure for Node 1 to get 0.32 and for Node 2 we get 0.4082 so the gain for using attribute B as a query is the weighted average

$$.5 - \left(\frac{5}{12}(.32) + \frac{7}{12}(.4082)\right) = .5 - 0.3715 = 0.128$$

so the gain is higher if we use attribute B to split the parent node. Note that using B results in a smaller weighted average so you get a bigger gain which makes sense because it is a measure of the impurity.

What happens if we use nominal attributes instead of binary to split the records.

If, for example, we have 3 nominal attributes then there are three ways these can be split; two with two child nodes and one with 3 child nodes. For the multiway split (i.e., 3 child nodes) we simply use $k = 3$ in our formula for the gain.

Example Return to our example of the decision tree for deciding whether an individual will default on a loan and decide whether it is better to query (i) home owner or (ii) marital status (assuming 2 child nodes) based on the data given. Use Gini measure. Assume class 1 is "no default."

The parent nodes consists of 10 records 7 of which did not default on the loan so the Gini measure is $1 - .7^2 - .3^2 = 0.420$.

If we use query (i) (home owner) then Node 1 has 3 records all of class 1 and none of class 2 so its Gini measure is 0. Node 2 has 4 records in class 1 and 3 in class 2 so its Gini measure is $1 - (4/7)^2 - (3/7)^2 = 0.4898$. So the gain is 0.42 -(0 +.7(.4898))=0.0771.

If we use query (ii) (marital status) then Node 1 has 4 records all of class 1 so its Gini measure is 0. Node 2 has 3 records in class 1 and 3 in class 2 so its Gini

is 0.5. Thus its gain is 0.42-(0 +.5(.6))=0.12. Thus query (ii) is better by this measure.

Example Here's a canonical example from classification we can investigate. Suppose we have collected the following attributes which we classify as binary, nominal, ordinal, continuous, etc.

ATTRIBUTE	POSSIBLE OUTCOMES
outlook	sunny, overcast, rain (nominal)
temperature	continuous
humidity	continuous
windy	true/false (binary)

Our goal is to predict whether a game (such as tennis) will be played. We use the following training data which consists of 14 records to build our model.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
overcast	83	78	false	Play
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
overcast	64	65	true	Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
rain	75	80	false	Play
sunny	75	70	true	Play
overcast	72	90	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play

Our goal is to determine which decision tree gives the largest information gain using the entropy (randomness) measure. We begin by deciding which attribute

to test first.

1. We start with choosing the outlook as the root node.

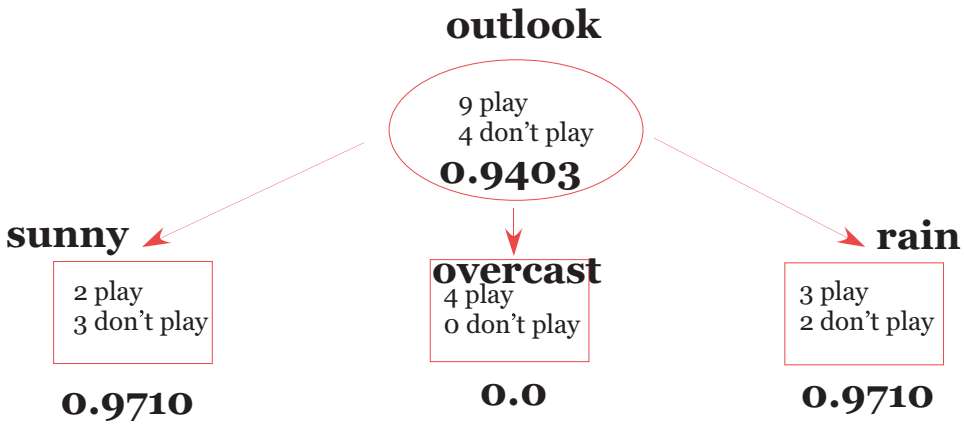
Out of the 14 plays 9 play and 5 don't so we compute the Entropy measure to get

$$-\big((9/14)\log_2(9/14) + (5/14)\log_2(5/14)\big) = 0.9403$$

Similarly the sunny outlook as 5 records, 2 of which play so

$$-\big(.4\log_2 .4 + .6\log_2 .6\big) = 0.9710$$

The overcast child node is homogeneous and so its measure is 0. The rain node has 3 records which play so it has the same measure as the sunny node. This is illustrated below.

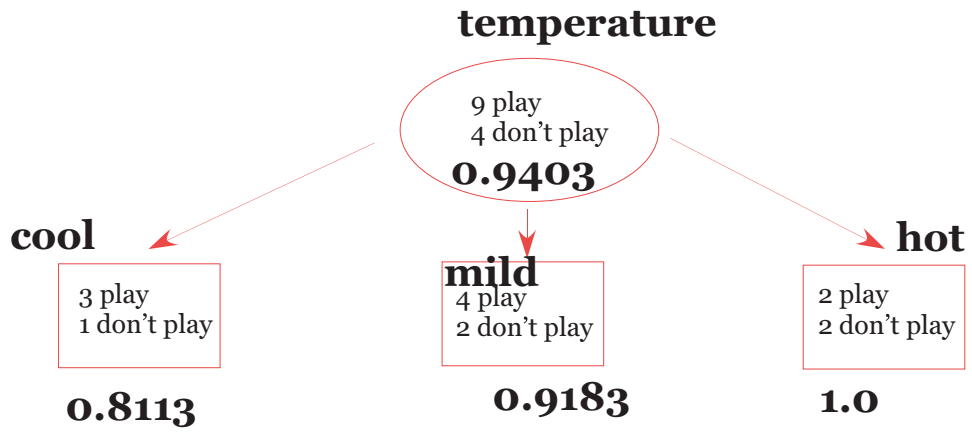


The gain in choosing this is determined by 0.9403 (parent measure) minus a weighted average of the three child nodes, i.e.,

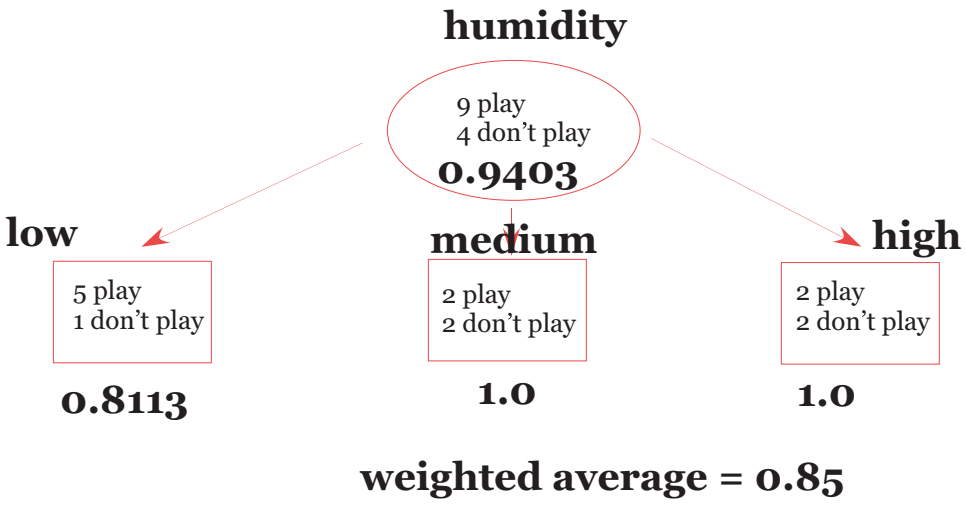
$$\text{information gain} = 0.9403 - \left[\frac{5}{14}.9710 + 0 + \frac{5}{14}.9710\right] = 0.9403 - 0.6929 = 0.2474$$

2. Now we start with the temperature as the root node and compute its gain. For simplicity we break the temperature into cool for temperatures below 70°, mild for temperatures $\geq 70°$ but $< 80°$ and hot for temperatures $\geq 80°$. The Entropy measure for each is shown in the figure and the weighted average for the three child nodes is 0.9111 so the gain is 0.0292 which is less than choosing the outlook as the parent node. We really didn't have to compute the gain, because

the measure for the child nodes was larger than the previous case so it resulted in a smaller gain.



3. Now we start with the humidity as the root node and compute its gain. We divide the humidity as low when it is ≤ 75, medium when it is between 75 and 90 and high for ≥ 90. The measures are given in the figure below and because the weighted average of the measure for the child nodes is 0.85 it is still larger than when we chose outlook as the parent node so the gain is less.



4. Lastly we choose windy as our first attribute to test. This is binary so we only have two child nodes. There are 6 windy records and it was evenly split between play and no play. For the remaining 8 records there were 6 plays and 2 no plays. Clearly the windy node has measure 1.0 and we compute the not windy node measure as 0.8113 so the weighted average is 0.8922 which results in a lower gain.

Consequently we choose outlook as the choice of the first attribute to test. Now we don't have to subdivide the overcast child node because it is homogeneous (pure) but the other two we need to divide. So if we take the sunny node then

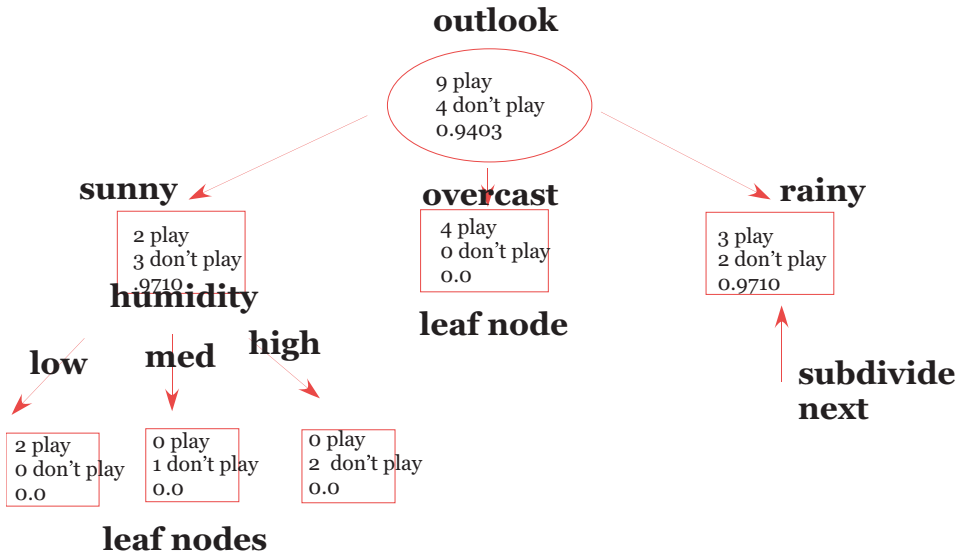
we have to decide whether to test first for temperature, humidity or windy. Then we have to do the same thing for the rainy node.

Here are the 5 sunny records.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
sunny	75	70	true	Play

We tabulate our results for the three remaining attributes below. Clearly the best choice is humidity because it results in all homogeneous child nodes (entropy measure $= 0.0$) so we don't even have to determine the weighted averages to determine the gain.

TEMPERATURE				HUMIDITY				WINDY			
	Play	Don't Play	Measure		Play	Don't Play	Measure		Play	Don't Play	Measure
cool	1	0	0.0	low	2	0	0.0	true	2	1	0.9183
mild	1	1	1.0	med	0	1	0.0	false	0	2	0.0
hot	0	2	0.0	high	0	2	0.0				



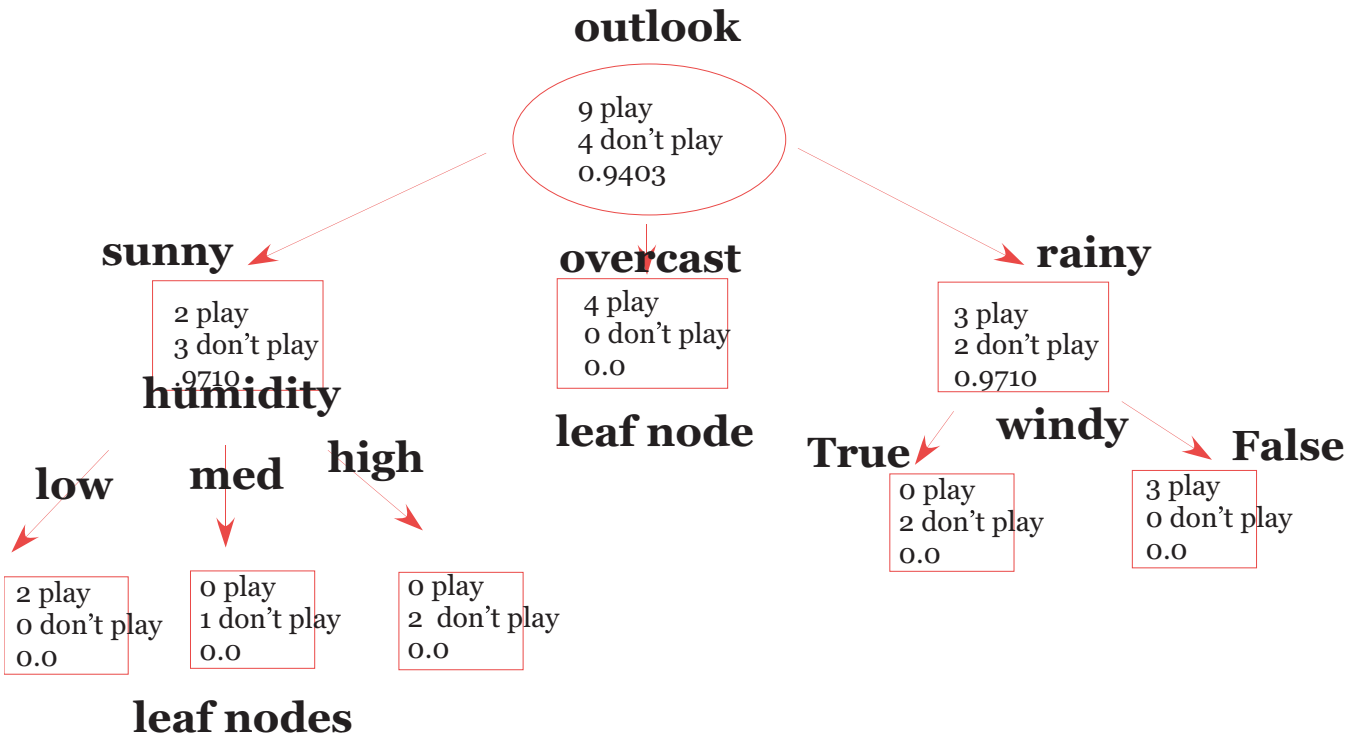
Here are the 5 rainy records which we want to determine how to subdivide. As before we compute the weighted average of our measure for the child nodes and

choose the smallest because it will result in the largest gain.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	71	80	true	Don't Play

TEMPERATURE				HUMIDITY				WINDY			
	Play	Don't Play	Measure		Play	Don't Play	Measure		Play	Don't Play	Measure
cool	1	1	1.0	low	0	1	0.0	true	0	2	0.0
mild	2	1	0.9183	med	2	1	0.9183	false	3	0	0.0
hot	0	0	0.0	high	0	0	0.0				

Once again we see that windy is the best choice because it results in all the nodes being homogeneous. Our final decision tree using our greedy algorithm with the entropy measure is given below.



Now suppose we have the following two new records to classify using our decision tree. What do you conclude?

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
rain	66	94	true	??
sunny	76	81	false	??

Other Classification Techniques

There are a multitude of classification techniques other than decision trees. We will only briefly look at a few of them due to time constraints.

- Rule-based classifiers
- Nearest neighbor classifiers
- Least Mean squares classifiers
- Bayesian classifiers
- Artificial Neural Networks
- Support Vector Machine
- Ensemble methods

Rule-based Classifier

In Rule-based classification we separate our data records using rules in the form of if – then constructs.

We will use the notation $\wedge$ for "and" and $\vee$ for "or".

To see how this classification technique works, assume that we have a training set of data on vertebrate which has the following attributes and their possible outcomes.

ATTRIBUTE	POSSIBLE OUTCOMES
body temperature	warm- or cold-blooded
skin cover	hair, scales, feathers, quills, fur, none
gives birth	yes/no
aquatic	yes/no/semi
aerial	yes/no
has legs	yes/no
hibernates	yes/no

Our class labels are

mammals, birds, reptiles, fishes, amphibians

From our training set (or from a biology course) we could develop the following rule set $\{r_1, r_2, r_3, r_4, r_5\}$ to classifiy each of the five labels.

$$r_1 : (\text{gives birth} = \text{no}) \wedge (\text{aerial=yes}) \implies \text{bird}$$

$$r_2 : (\text{gives birth} = \text{no}) \wedge (\text{aquatic=yes}) \implies \text{fish}$$

$$r_3 : (\text{gives birth} = \text{yes}) \wedge (\text{body temperature} = \text{warm-blooded}) \implies \text{mammals}$$

$$r_4 : (\text{gives birth} = \text{no}) \wedge (\text{aerial=no}) \implies \text{reptile}$$

$$r_5 : (\text{aquatic} = \text{semi}) \implies \text{amphibian}$$

Now consider two new records which we want to classify

NAME	BODY TEMP	SKIN COVER	GIVES BIRTH	AQUATIC	AERIAL	LEGS	HIBERNATES
grizzly bear	warm	fur	yes	no	yes	yes	yes
turtle	cold	scales	no	semi	no	yes	no
flightless	warm	feathers	no	no	no	yes	no
cormorant							
guppy	cold	scales	ues	yes	no	no	no

Now the grizzly bears does not satisfy the conditions of r_1 or r_2 but r_3 is triggered and it is classified as a mammal.

The turtle triggers r_4 and r_5 but the conclusions of these rules are contradictory so it can't be classified with this rule set.

The flightless cormorant triggers rule 4 so it is incorrectly classified as a reptile.

The last record does not trigger any of the five rules.

We say the rules in a rule set are mutually exclusive if no two rules are triggered by the same record. Thus our rule set above is not mutually exclusive because the record for turtle triggers two rules with contradictory classifications.

We say the rules in a rule set are exhaustive if there is a rule for each combination of the attribute set. Thus our rule set is not exhaustive because it failed to consider the combination of attributes that the guppy has.

Example Can you write a set of mutually exclusive and exhaustive rules which

classify vertebrates as mammals or non-mammals?

Usually one describes the quality of a classification rule by a measure of its accuracy and coverage. If we have a set of records $\mathcal{D}$ and a rule r which classifies data as class y then its coverage is just the fraction of $\mathcal{D}$ that the rule triggers, say $\mathcal{D}_r/\mathcal{D}$. The accuracy or confidence level is just the fraction of records that it classifies correctly, i.e.,

$$\text{accuracy} = \frac{\text{records in } \mathcal{D}_r \text{ which are of class } y}{\mathcal{D}_r}$$

Ordered Rules

Just like when you program if-then constructs the ordering of classification rules can be important. For example, if one rule is expected to have a much higher coverage than the others, we might want to order it first; there are other choices for ranking the rules too. So if we have an ordered set we could avoid the problem we encountered classifying the turtle record. Because we order the rules by some priority we classify a record based on the first rule that it triggers. Unfortunately, in our ordering of the rules we would have classified the turtle record incorrectly

as a reptile so our ordering was not so good. Interchanging rules 4 and 5 would fix this problem but of course it could create others.

Unordered Rules

We could take a different approach and let a record trigger multiple rules. In this case we keep a count of each way we classify the record and then just go with the majority (if there is one). This approach is more computationally intensive because we have to check whether our record satisfies the conditions of each rule whereas in a set of ordered rules we only have to check until the first record is triggered.

Suppose now that we decide to order our rules. What should our strategy be?

If we group all the rules for one class together then we are using class-based ordering.

If we use some quality measure (like coverage) then it is just called rule-based ordering. Remember that when we use ordered rules then when you are at, say

rule r_{10}, you are assuming that the negations of the previous conditions are true so it can often be a bit confusing to interpret a rule.

Here are a few rules in a possible set of class-based ordering for our vertebrate classification. A rule-based ordering set could be any combination of these where we don't group all of our outcomes together.

Class-based ordering

r_1 (aerial=yes) $\wedge$ (skin cover = feathers) $\Longrightarrow$ birds

r_2 (body temp=warm) $\wedge$ (gives birth =no) $\Longrightarrow$ bird

r_3 (body temp=warm) $\wedge$ (gives birth =yes) $\Longrightarrow$ mammal

r_4 (aquatic=semi) $\wedge$ $\Longrightarrow$ amphibian

$\vdots$

Note that this set of rules correctly classifies the flightless cormorant record because it triggers rule r_2 (but not r_1) and it also correctly classifies the turtle

record because it triggers r_4 and not the previous three rules. However, these rules can't identify the guppy but we haven't added any yet to classify a fish.

Of course the critical question we have not addressed yet is how to build a rule-based classifier. We have to pose our rules in an intelligent manner so that the rules identify key relationships between the attributes of the data and the class label.

Two broad approaches are usually considered. We can look at the training data and try to develop our rules based on it (direct methods) or we can use the results of another classification model such as a decision tree to develop our rules (indirect methods). We consider a direct method here.

Sequential Covering Algorithm

This is a Greedy algorithm which has the strategy that it learns one rule, remove the records it classifies and then repeats the process. One has to specify what criterion to use to order the class labels. So suppose we order our class labels as $\{y_1, y_2, \ldots\}$ and want to develop a rule which classifies y_1 that covers the

training set. All records in the training set that are labeled as class y_1 are considered positive examples and those not labeled as class y_1 are considered negative examples. We seek a rule which covers a majority of the positive examples and none/few of the negative examples.

So basically we just need a routine to learn one rule. Because after we learn this rule we simply remove all records from our training set that are labeled as y_1 and repeat the process again with y_2.

Learn one rule function

Our approach is to "grow" our rule using a greedy strategy. We can either take the approach of starting with a guess for a general rule and then adding conditions to make it more specific or the converse of starting with a specific rule and then pruning it to get a more general rule.

Let's take the general to specific approach for finding a rule to classify mammals. The most general rule is

$$(\quad) \Longrightarrow \text{mammal}$$

which every record satisfies because there is no condition to check. Assume that our training set is given as follows (taken from Tan, et al, Intro to Data Mining).

NAME	BODY TEMP	SKIN COVER	GIVES BIRTH	AQUATIC	AERIAL	LEGS	HIBER-NATES	CLASS
human	warm	hair	yes	no	no	yes	no	mammals
python	cold	scales	no	no	no	no	yes	reptile
salmon	cold	scales	no	yes	no	no	no	fish
whale	warm	hair	yes	yes	no	no	no	mammal
frog	cold	none	no	semi	no	yes	yes	amphibian
k.dragon	cold	scale	no	no	no	yes	no	reptile
bat	warm	hair	yes	no	yes	yes	yes	mammal
robin	warm	feathers	no	no	yes	yes	no	bird
cat	warm	fur	yes	no	no	yes	no	mammals
guppy	cold	scales	yes	yes	no	no	no	fish
alligator	cold	scales	no	semi	no	yes	no	reptile
penguin	warm	feathers	no	semi	no	yes	no	bird
porcupine	warm	quills	yes	no	no	yes	yes	mammal
eel	cold	scales	no	yes	no	no	no	fish
newt	cold	none	no	semi	no	yes	yes	amphibian

So now we want to add a condition to the rule. We look at testing each of the 6 attributes for the 15 records in our training set.

BODY TEMPERATURE = WARM has 7 records satisfying this where 5 are labeled as mammals

BODY TEMPERATURE = COLD has 8 records satisfying this where 0 are labeled as mammals

SKIN COVER = HAIR has 3 records satisfying this with all 3 labeled as mammals

SKIN COVER = QUILLS results in 1 record satisfying it and it is labeled mammal

SKIN COVER = FUR results in 1 record satisfying it and it is labeled mammal

SKIN COVER = SCALES results in 6 record satisfying it and none are labeled mammal

SKIN COVER = FEATHERS results in 2 record satisfying it and none are labeled

mammal

SKIN COVER = NONE results in 2 records satisfying it and none are labeled mammal;

GIVES BIRTH=YES has 6 records satisfying where 5 are labeled as mammals

GIVES BIRTH=NO has 9 records satisfying this but none are labeled as mammals

AQUATIC=YES has 4 records satisfying this with 1 labeled as mammal

AQUATIC=SEMI has 4 records satisfying none are labeled as mammals

AQUATIC=NO has 7 records satisfying this where 4 are labeled as mammals

AERIAL=YES has 2 records satisfying this where 1 is labeled as mammals

AERIAL=NO has 13 records satisfying this where 5 are labeled as mammals

HAS LEGS=YES has 10 records satisfying this where 4 are labeled as mammals

HAS LEGS=NO has 5 records satisfying this where 1 is labeled as mammals

HIBERNATES=YES has 5 records satisfying this where 2 are labeled as mammals

HIBERNATES=NO has 10 records satisfying this where 3 are labeled as mammals

Now let's compute the coverage and accuracy of each which had outcomes labeled as mammal because this is what we are trying to label.

attribute	coverage	accuracy
BODY TEMPERATURE = WARM	7/15	5/7
SKIN COVER = HAIR	3/15	1
SKIN COVER = QUILL	1/15	1
SKIN COVER = FUR	1/15	1
GIVES BIRTH=YES	6/15	5/6
AQUATIC=YES	4/15	1/4
AQUATIC=NO	7/15	4/7
AERIAL=YES	2/15	1/2
AERIAL=NO	13/15	5/13
HAS LEGS=YES	10/15	4/10
HAS LEGS=NO	5/15	1/5
HIBERNATES=YES	5/15	2/5
HIBERNATES=NO	10/15	3/10

Clearly we don't want to go strictly on accuracy because, for example, SKIN COVER = QUILL is completely accurate for our training set but its coverage is only one out of 15 records. If we look at a combination of the coverage and accuracy then the two contenders seem to be (i) BODY TEMPERATURE

= WARM and (ii) GIVES BIRTH=YES. We could choose either based on the criteria we implement. We will discuss criteria shortly.

Once we make our decision we add this to our rule, say we choose BODY TEMPERATURE = WARM. Now we need to test the remaining 6 attributes to make our next choice. Without going through all the possible outcomes let's just assume that GIVES BIRTH=YES results in the best because its coverage and accuracy can be easily determined as 5/7 and 1.0, respectively. To see this note that there are 7 records that satisfy BODY TEMPERATURE = WARM. When we query GIVES BIRTH=YES we get 5/7 coverage with all accurately classified. Thus the rule

$$(\text{ BODY TEMPERATURE} = \text{WARM }) \wedge (\text{ GIVES BIRTH=YES }) \implies \text{mammal}$$

is our complete rule to classify mammals. We then remove the 5 records in the training set that are labeled mammal and choose our next label y_2 and begin again.

If we take a specific to general approach then we start with the rule

$($ BODY TEMPERATURE $=$ WARM $) \wedge$ (SKIN COVER $=$ HAIR $) \wedge ($ GIVES BIRTH$=$YES $) \wedge ($ AQUATIC$=$NO $) \wedge ($ AERIAL $=$ NO $) \wedge ($ HAS LEGS$=$YES $) \wedge ($ HIBERNATES $= 0$ $) \implies$ mammal

There is only one record (human) in the training set that has all of these attributes so clearly we need to remove some. The procedure is analogous to before.

What criteria should we use to add a rule?

We have seen that accuracy is not enough, because, e.g., we had complete accuracy for the rule SKIN COVER $=$ QUILL but there was only 1 positive example of this in our training set. Coverage alone is not enough because, e.g., HIBERNATES$=$NO had 10 positive examples in the training set but was only 30% accurate. Here are a couple of commonly used evaluation metrics.

Let n denote the number of records which satisfy the condition of the rule, n_+ denote the number of positive records (i.e., the ones for which the outcome is

true) and let k denote the number of classes. Then

$$\text{Laplace} = \frac{n_+ + 1}{n + k}$$

Let's look at this measure for two of our examples above.; here $k = 5$ (mammals, fish, reptiles, amphibians, bird).

SKIN COVER = QUILL Laplace $= \frac{2}{20} = .1$

HIBERNATES=NO Laplace $= \frac{4}{15} = .266$

Now let's compare two attributes and compare. Recall that BODY TEMPERATURE = WARM had a coverage of 7/15 and an accuracy of 5/7

SKIN COVER = QUILL has a coverage of 1/15 and an accuracy of 1.0

The second one has a better accuracy but lower coverage. Let's compute the Laplace measure for each.

BODY TEMPERATURE = WARM Laplace $= \frac{5+1}{7+5} = .5$

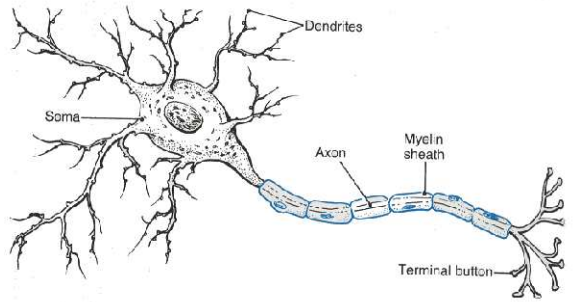
SKIN COVER = QUILL Laplace $= \frac{1+1}{1+5} = .3333$

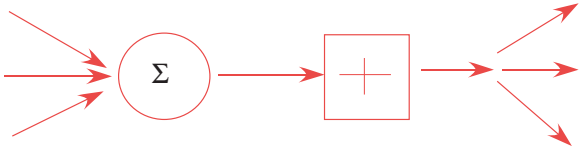
If we compare this with the accuracy the second test has a much smaller value of its Laplace measure than its accuracy so this says the accuracy of 1.0 was spurious because the test didn't has enough positive records.

Clearly one can think of many other metrics or combinations thereof to use.

Neural Networks (NN)

This idea was inspired by attempts to simulate a biological neural system where neurons are linked together via axons which are used to transmit nerve impulses from one neuron to another. Neurons are connected to axons of other neurons via dendrites. The connection between the dendrite and axon is called a synapse and we learn by changing the strength of this connection. It is estimated that the human brain has 10^{10} neurons each of which has thousands of connectors.





Neural Networks have seen an explosion of interest over the last few years, and are being applied across a range of problems as diverse as finance, medicine, engineering, geology and physics. Neural Networks are used where problems involve prediction, classification or control.

For many years linear modeling (like linear regression) was the commonly used technique in most modeling because linear models are simple to use and have well-known optimization strategies. Of course if a linear approximation is not valid (which is frequently the case) the models are not representative of the data.

Neural networks are very sophisticated modeling techniques capable of modeling complex functions. In particular, neural networks can be nonlinear. Neural networks also keep in check the "curse of dimensionality".

Neural networks learn by example. The neural network user gathers representative data, and then invokes training algorithms to automatically learn the structure of the data. Although the user does need to have some heuristic knowledge of how to select and prepare data, how to select an appropriate neural network, and how to interpret the results, the level of user knowledge needed to successfully

apply neural networks is much lower than would be the case using some other methods.

Some Applications of Neural Networks

- Detection of medical phenomena A variety of health-related indices (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a complex (e.g., nonlinear and interactive) combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed.
- Stock market prediction Fluctuations of stock prices and stock indices are complex and multidimensional, but in some circumstances at least partially-deterministic phenomenon. Neural networks are being used by many technical analysts to make predictions about stock prices based upon a large number of factors such as past performance of other stocks and various economic indicators.

- Credit assignment A variety of pieces of information are usually known about an applicant for a loan. For instance, the applicant's age, education, occupation, and many other facts may be available. After training a neural network on historical data, neural network analysis can identify the most relevant characteristics and use those to classify applicants as good or bad credit risks.
- Monitoring the condition of machinery Neural networks can be instrumental in cutting costs by bringing additional expertise to scheduling the preventive maintenance of machines. A neural network can be trained to distinguish between the sounds a machine makes when it is running normally ("false alarms") versus when it is on the verge of a problem. After this training period, the expertise of the network can be used to warn a technician of an upcoming breakdown, before it occurs and causes costly unforeseen down-time.

Now to describe neural nets we want to capture the essence of biological neural systems so we define an artificial neuron as follows:

- It receives a number of inputs (either from original data, or from the output

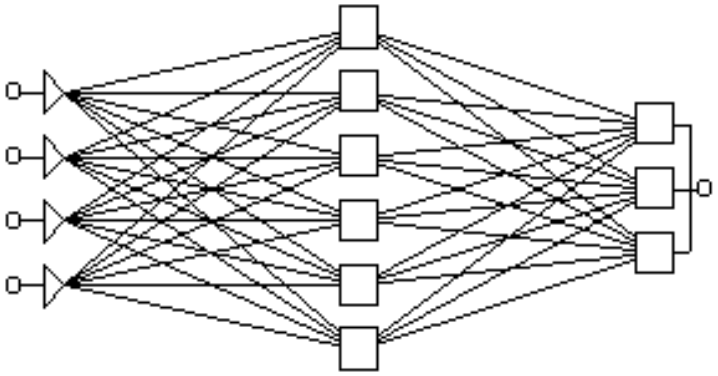
of other neurons in the network).

- Each input comes via a connection that has a strength (i.e., a weight); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to determine the activation of the neuron.
- The activation signal is passed through an activation function (also known as a transfer function) to produce the output of the neuron.

Now we have to decide how to connect the neurons.

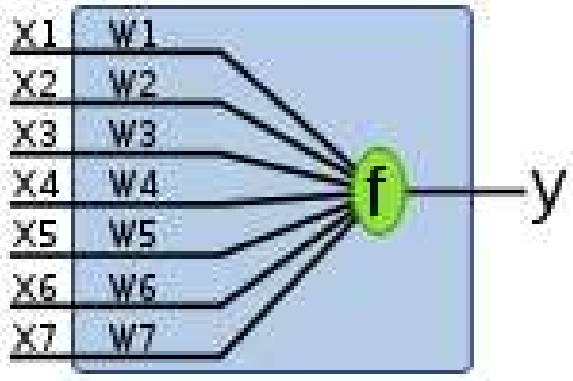
- If a network is to be of any use, there must be inputs (which carry the values of variables of interest) and outputs (which form predictions). Inputs and outputs correspond to sensory and motor nerves such as those coming from the eyes and leading to the hands.
- A simple network has a feedforward structure: signals flow from inputs, forward through any hidden units, eventually reaching the output units. Such a structure has stable behavior.
- A typical feedforward network has neurons arranged in a distinct layered

topology. The input layer is not really neural at all: these units simply serve to introduce the values of the input variables. The hidden and output layer neurons are each connected to all of the units in the preceding layer.



Perceptron - A linear classifier neural network

Let's look at a very simple model of a neural net called a perceptron and see how it is used. It was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt (according to Wikipedia). It can be seen as the simplest kind of feedforward neural network and is a linear classifier.



The perceptron is a binary classifier which maps its inputs $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\}$ to an output value $f(\vec{x}_i)$ which is a single binary value. Each input has an associated weight w_i so we have a weight vector $\vec{w}$ and the neuron has a threshold θ. Our goal is to find the weights so that the input data is classified. We determine

whether the neuron is activated or not based upon the value of f which is assigned by

$$f(\vec{x}) = \begin{cases} 1 & \vec{w}^T\vec{x} - \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

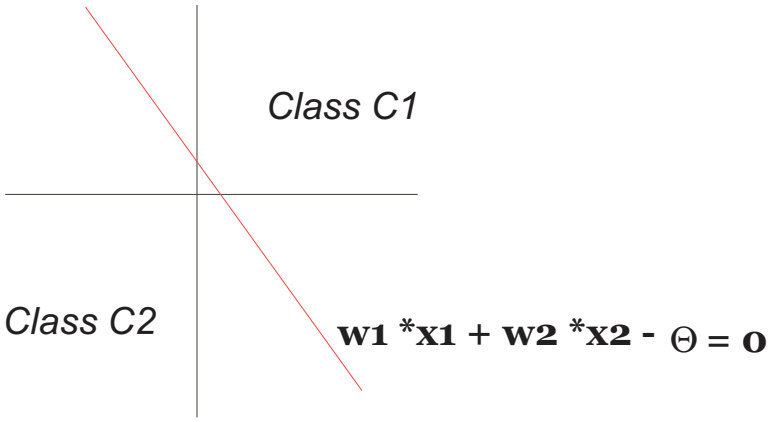
If $f(\vec{x}) > 0$ we say that the neuron has activated or fired. The scalar product $\vec{w}^T\vec{x}$ computes a weighted sum for the inputs and in order for the neuron to become activated this weighted sum must be larger than the threshold θ.

The value of $f(\vec{x})$ is either 0 or 1 so it can only be used in a binary classification problem; for example mammal or non-mammal.

So that we can draw some pictures, let's consider a problem in two dimensions. Our decision boundary is governed by the line

$$w_1x_1 + w_2x_2 = \theta$$

A point that lies on or above the line is assigned to class $\mathcal{C}_1$ and below the line is assigned to class $\mathcal{C}_2$. So if we have a random point $\vec{x} = (a, b)$ then if $w_1a + w_2b \geq \theta$ it is of class $\mathcal{C}_1$ and if it is $< \theta$ it is in $\mathcal{C}_2$. Another way to say this is that $\vec{w}^T\vec{x} - \theta \geq 0$ implies $\vec{x} \in \mathcal{C}_1$ which is just what $f(\vec{x})$ calculates.



In general, we suppose that the input variables of the (single-layer) perceptron originate from two linearly separable classes that fall on the opposite sides of some hyperplane $\vec{w}^T\vec{x} - \theta = 0$. Of course not all data will be linearly separable.

Suppose our training set is composed of two disjoint sets of records X_1 and X_2 where members of X_1 are in $\mathcal{C}_1$ and members of X_2 are in $\mathcal{C}_2$. Our goal is the adjust the weight factors so that the classes are separable. For each input $\vec{x}_i$ in our training set we know which class it is in so our training data is of the form $(\vec{x}_i, y_i)$ where y_i is the class label.

How do we compute the weights so that the training data is separated?

We employ an iterative method which alters the weights to reinforce correct decisions and discourage wrong decisions, hence reducing the error. For example, if $f(\vec{x}_i) > 0$ and $\vec{x}_i$ is in class $\mathcal{C}_1$ then we are happy with the weights so we don't change them but if $\vec{x}_i$ is in class $\mathcal{C}_2$ then we need to adjust our weights because $f(\vec{x}_i)$ should be < 0 for inputs in $\mathcal{C}_2$.

Geometrically we can view $\vec{x}_i$ and $\vec{w}$ as vectors emanating from the origin and we seek $\vec{w}$ which has a positive projection with all training examples in Class $\mathcal{C}_1$ and a negative projection (i.e., inner product) with all training examples in Calss $\mathcal{C}_2$.

Of course we need a parameter to decide how much to adjust the weights each time. We will set a constant learning rate η where $0 < \eta < 1$. One can use an adaptive learning rate where we update it each iteration.

Perceptron Algorithm

Assume we are given a set of m training records $\vec{x}_i$, $i = 1, \ldots, m$ and class labels y_i which indicate whether they are in class $\mathcal{C}_1$ or $\mathcal{C}_2$; we set $y_i = 1$ if $\vec{x}_i \in \mathcal{C}_1$ and $y_i = -1$ if $\vec{x}_i \in \mathcal{C}_2$. So our input data is pairs of the form $(\vec{x}_i, y_i)$. Let $\vec{w}$ denote the value of the weights and w_i the weight corresponding to input node i.

For the algorithm we will set up vectors of length $n+1$ for the weights and input data so that when we take the inner product we will be computing $\vec{w}^T\vec{x} - \theta$. To this end, set $\vec{w} = (\theta, w_1, w_2, \ldots, w_n)$ and each input data $\vec{x} = (-1, x_1, x_2, \ldots, x_n)$. Let $\Delta\vec{w}$ represent the amount we will change the weights by at each step.

Initialize weights Set $\vec{w} = \texttt{rand}(-.05, .05)$; set learning parameter η

`while` convergence not attained

$\Delta\vec{w} = 0$

`for each` $(\vec{x}_i, y_i)|_{i=1}^{m}$ `do`

compute $(\vec{w}^k)^T \vec{x}(k)$

determine $f(\vec{x}_i)$ % activate neuron or not

$\delta_i = y_i - f(\vec{x}_i)$

update the change in the weights $\Delta\vec{w} = \Delta\vec{w} + \eta\delta_i\vec{x}_i$

end for loop

check for convergence

end while loop

If $\delta_i \neq 0$ then we update the weights. Let's look at the possible values of δ_i for our code.

$$\vec{w}^T\vec{x}_i > 0 \implies f = 1 \quad \begin{cases} \text{if } \vec{x}_i \in \mathcal{C}_1 & y_i = 1 \implies \delta_i = 0 \\ \text{if } \vec{x}_i \in \mathcal{C}_2 & y_i = -1 \implies \delta_i = -1 \end{cases}$$

$$\vec{w}^T\vec{x}_i < 0 \implies f = 0 \quad \begin{cases} \text{if } \vec{x}_i \in \mathcal{C}_1 & y_i = 1 \implies \delta_i = -1 \\ \text{if } \vec{x}_i \in \mathcal{C}_2 & d = 0 \implies \delta_i = 0 \end{cases}$$

It is important to realize that there are infinitely many hyperplanes that separate the linearly separable data. The Perceptron Algorithm has the (possible) disadvantage that it stops moving the line (in 2-D) as soon as it finds any line that separates the data. Other approaches may find a "better" line. We will look at some other approaches.

Example Use the Perceptron Algorithm to separate the points

$$\mathcal{C}_1 : \quad (2, -10), (4, 2), (1, 1), (2, 6)$$

$$\mathcal{C}_2 : \quad (2, -2), (-1, 0), (0, 4)$$

In the case where we set $\eta = 0.5$ we get the line $y = 6x - .5$ (approximately). When we set $\eta = 0.25$ we get the same line except our weights are cut by 0.5.

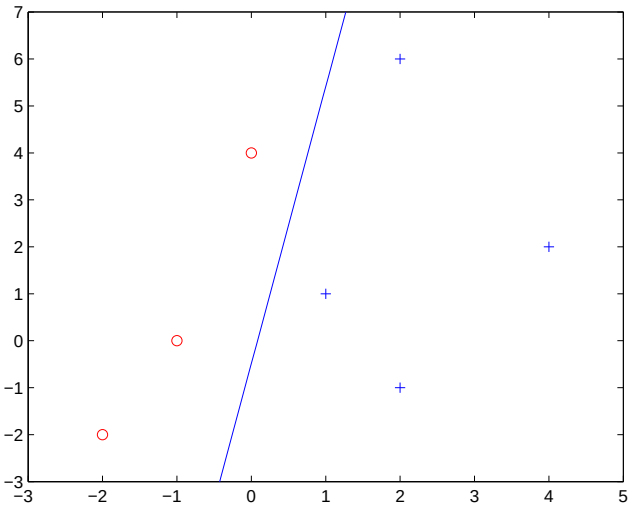
If we modify the data slightly so that a member from both classes has the same

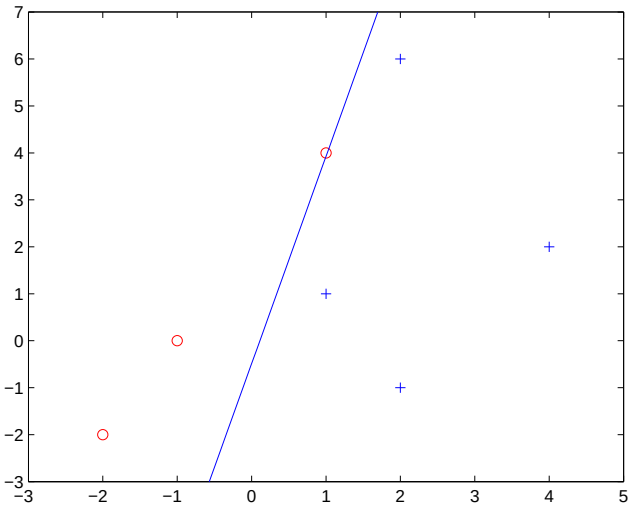
x coordinate, i.e.,

$$\mathcal{C}_1 : \quad (2, -10), (4, 2), (1, 1), (2, 6)$$

$$\mathcal{C}_2 : \quad (2, -2), (-1, 0), (1, 4)$$

we get the result illustrated where $(1, 4)$ is slightly to the left of the line.



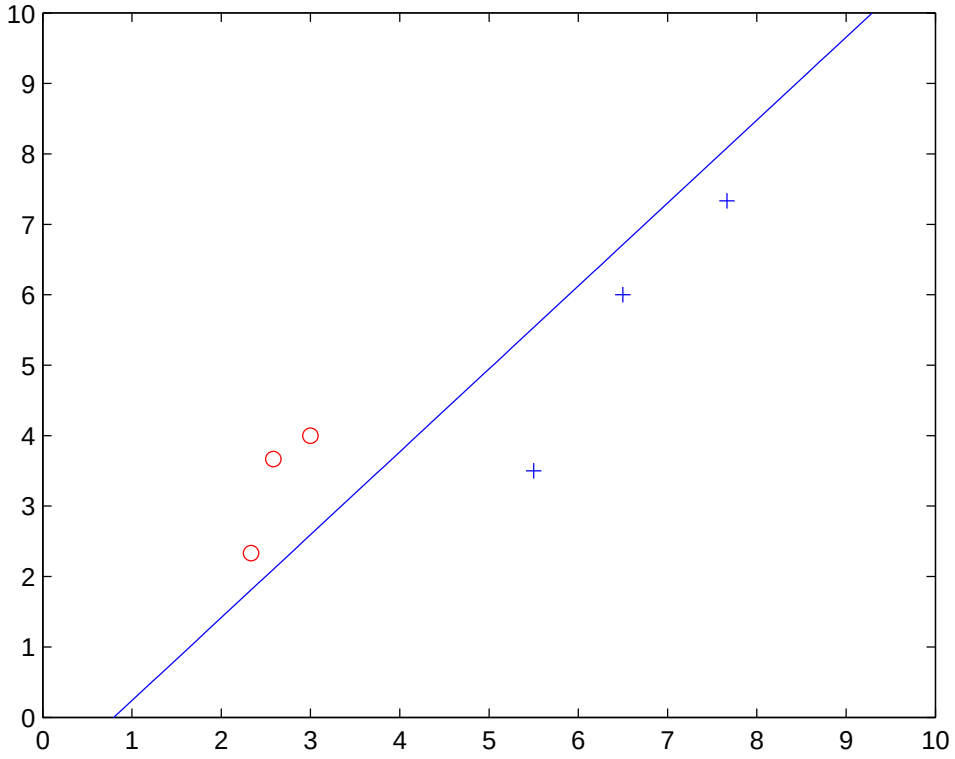


Example Suppose we want to classify objects as either beds or chairs based upon their length and width. We can use this Perceptron Algorithm if the training data is linearly separable. Consider the following training classes.

$\mathcal{C}_1$ – beds: 66 in by 42 in ; 92 in by 88 in; 78 in by 72 in

$\mathcal{C}_2$ – chairs: 28 in by 28 in ; 31 in by 44 in; 36 in by 48 in

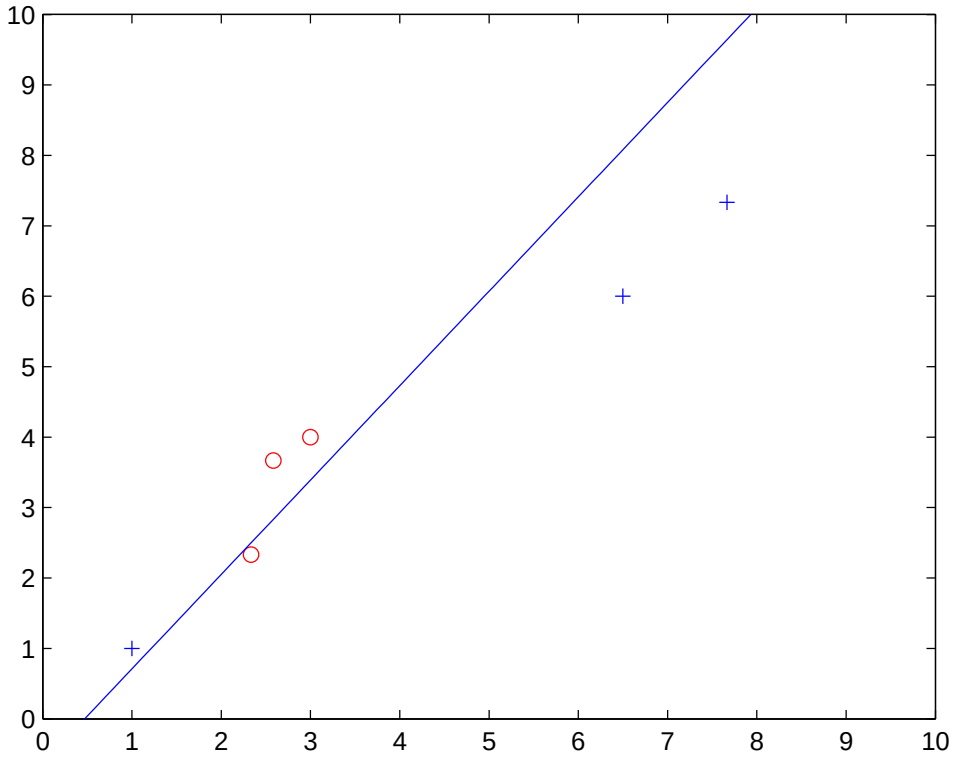
We can apply our algorithm as before to find the line that separates the data. Here we have converted the dimensions to feet.



If we modify the data so that it is not separable, i.e., we choose $\mathcal{C}_1$ as

$$\mathcal{C}_1 - \text{beds:} \quad \text{12 in by 12in ; 92 in by 88 in; 78 in by 72 in}$$

Then the algorithm can't converge. Here are the results after 25 iterations.



It is important to realize that there are infinitely many hyperplanes that separate

the linearly separable data. The Perceptron Algorithm has the (possible) disadvantage that it stops moving the line (in 2-D) as soon as it finds any line that separates the data. Other approaches may find a "better" line. We will look at some other approaches.

Examples of a Multilayer Perceptron

To see how we can use a perceptron in a more complicated example we consider an example from pattern recognition. Before we do this we will look at a simple case of firing a neuron. We won't be going into the details but you can get an idea how it could work. The reference for examples is Neural Networks by Christos Stergiou and Dimitrios Siganos.

The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained.

Take a collection of training patterns for a node, some of which cause it to fire (the "1" taught set of patterns) and others which prevent it from doing so (the "0" taught set). Then the patterns not in the collection cause the node to fire if, on comparison , they have more input elements in common with the "nearest"

pattern in the "1"-taught set than with the "nearest" pattern in the "0"-taught set. If there is a tie, then the pattern remains in the undefined state.

As an example, consider a 3-input neuron which is taught to output 1 when the input (X1,X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001. Before we apply the firing rule, we consider the following table.

X1:	0	0	0	0	1	1	1	1
X2:	0	0	1	1	0	0	1	1
X3:	0	1	0	1	0	1	0	1
OUT:	0	0	0/1	0/1	0/1	1	0/1	1

The four training inputs are listed in this table (e.g., if the input is 000 the output is 0 or if the input is 101 then the output is 1) but other combinations are listed with output either 0 or 1 – to be determined.

As an example of the way the firing rule is applied, take the input 010. It differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from

111 in 2 elements. Therefore, the 'nearest' pattern is 000 which belongs in the 0-taught set. Thus the firing rule requires that the neuron should not fire when the input is 001. On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1).

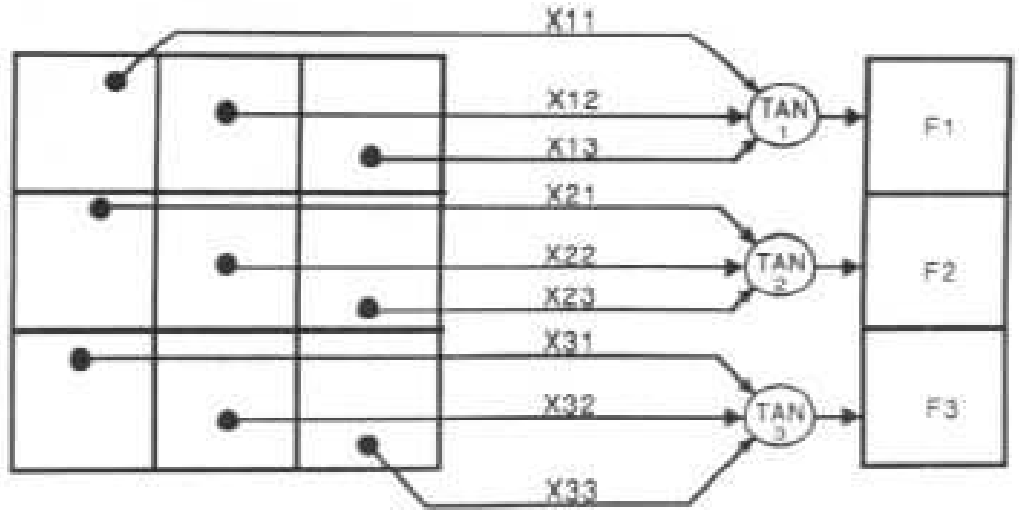
By applying the firing in every column the following truth table is obtained:

```
X1:  0 0 0 0 1 1 1 1
X2:  0 0 1 1 0 0 1 1
X3:  0 1 0 1 0 1 0 1
OUT: 0 0 0 0/1 0/1 1 1 1
```

The difference between the two truth tables is called the generalization of the neuron. Therefore the firing rule gives the neuron a sense of similarity and enables it to respond "sensibly" to patterns not seen during training.

An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward neural network that has been trained accordingly. During training, the network is trained to associate outputs with input patterns. When the network is used, it identifies the input

pattern and tries to output the associated output pattern. The power of neural networks comes to life when a pattern that has no output associated with it, is given as an input. In this case, the network gives the output that corresponds to a taught input pattern that is least different from the given pattern.



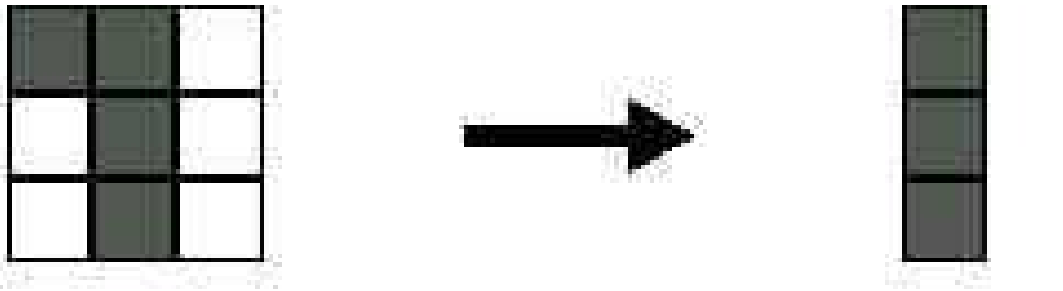
Assume that the network shown above is trained to recognise the patterns “T” and “H”. The associated patterns are all black and all white respectively as shown below.

If we represent black squares with 0 and white squares with 1 then the truth tables for the 3 neurons after generalization can be written as

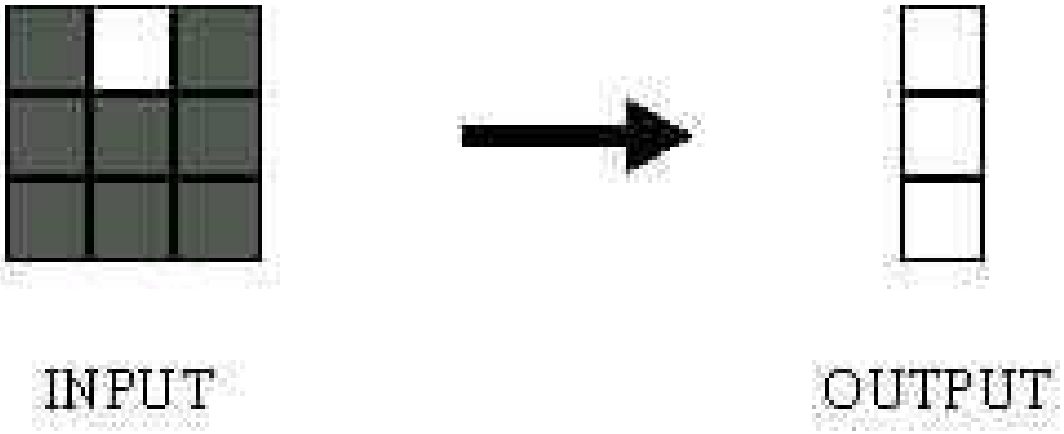
```
X11: 0 0 0 0 1 1 1 1
X12: 0 0 1 1 0 0 1 1
X13: 0 1 0 1 0 1 0 1
OUT: 0 0 1 1 0 0 1 1
 Top neuron

X21: 0  0  0  0   1  1  1  1
X22: 0  0  1  1   0  0  1  1
X23: 0  1  0  1   0  1  0  1
OUT: 1 0/1 1 0/1 0/1 0 0/1 0
 Middle neuron
```

```
X21: 0 0 0 0 1 1 1 1
X22: 0 0 1 1 0 0 1 1
X23: 0 1 0 1 0 1 0 1
OUT: 1 0 1 1 0 0 1 0
 Bottom neuron
```

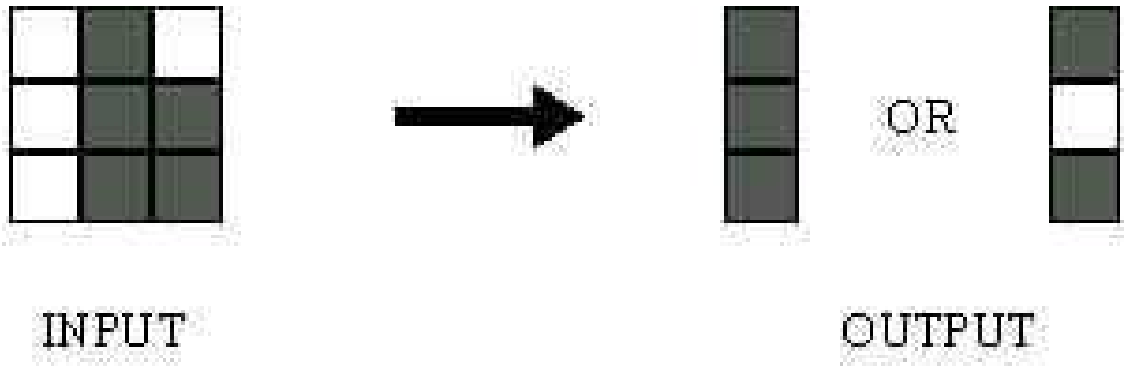
From the tables we can get the following associations:



In this case, it is obvious that the output should be all blacks since the input pattern is almost the same as the "T" pattern.



Here it is obvious that the output should be all whites since the input pattern is almost the same as the "H" pattern.



Here, the top row is 2 errors away from the a T and 3 from an H. So the top

output is black. The middle row is 1 error away from both T and H so the output is random. The bottom row is 1 error away from T and 2 away from H. Therefore the output is black. The total output of the network is still in favour of the T shape.

Least-Mean-Square (LMS) Neural Networks

We will look at another type of Neural Networks which consists of a single neuron and which assumes that the data is linearly separable.

This method is also known by the names delta rule and Widrow-Hoff rule and uses the Steepest Descent approach that we have already discussed. It was originally formulated for use in adaptive switching circuits.

Again assume we have input data (signals) $\vec{x}_i$, $i = 1, \ldots, m$ and there is a corresponding set of weights $\vec{w}$. The requirement now is to determine the optimum setting of the weights so we minimize the difference between the system output and some desired or target response y in a mean square sense.

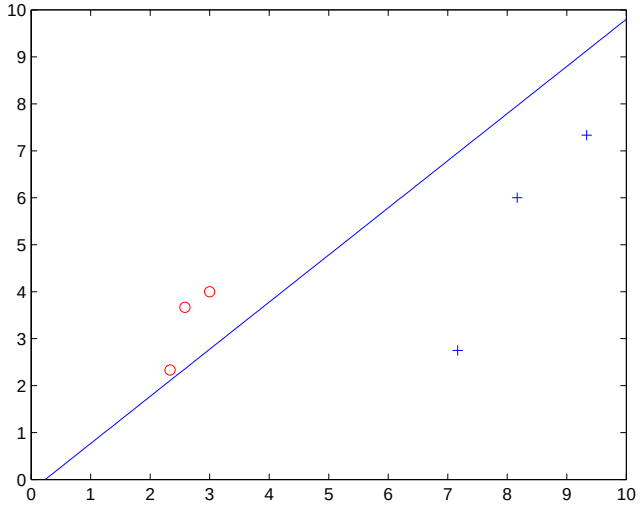
If we have m training examples $\vec{x}_i$ with corresponding labels y_i then the mean

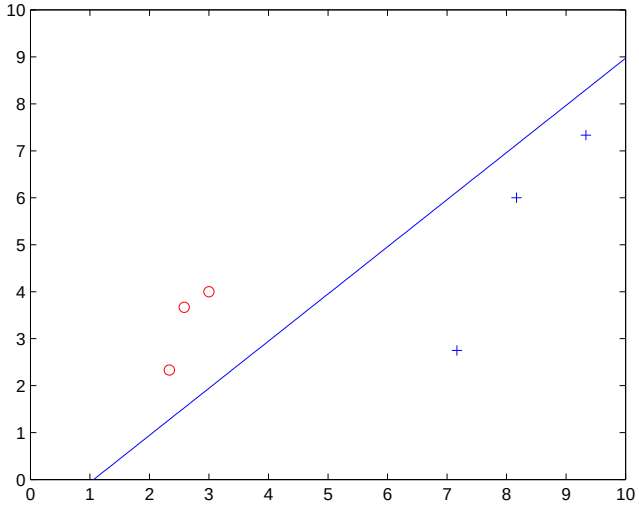
square error $\mathcal{E}$ is given by

$$\mathcal{E} = \frac{1}{m} \sum_{k=1}^{m} (\vec{w}^T \vec{x}_k - y_k)^2$$

The basic idea is to apply the Steepest Descent algorithm to find the weights that minimize this error. As before, we take a step in the direction of minus the gradient which in this case is $-\partial \mathcal{E} / \partial w_i$. The method proceeds in the standard way.

In many cases the LMS approach can get a better hyperplane separating the data. However, if the data has a data point that lies far from the other data in its class then this can adversely affect the LMS solution.


0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10

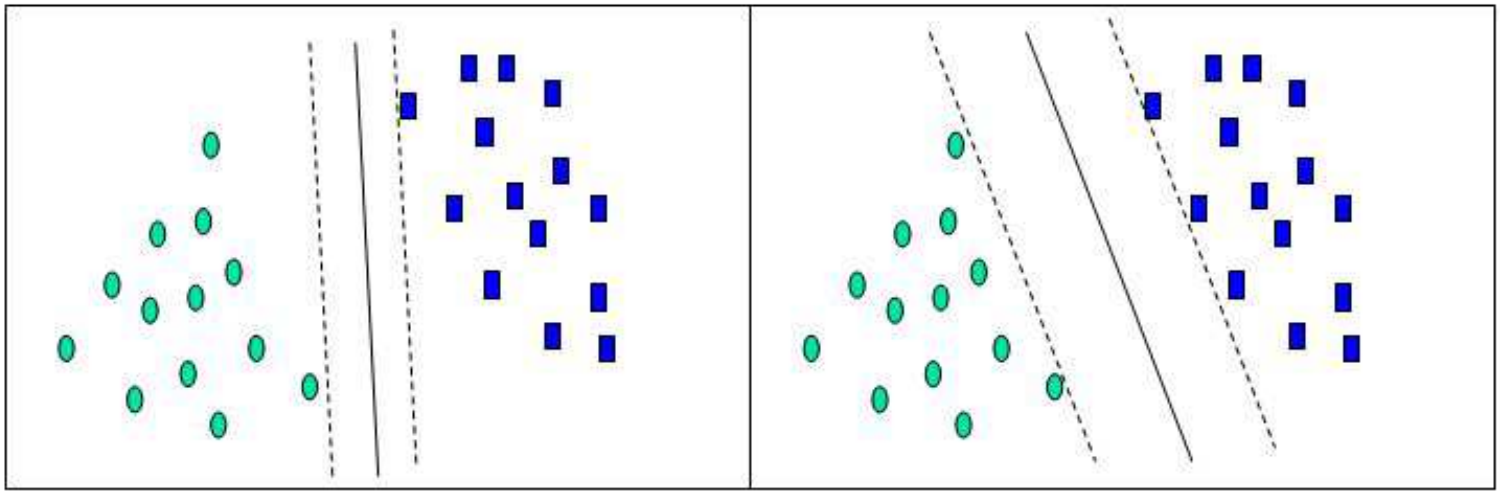

0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10

Support Vector Machine (SVM)

SVM performs classification by constructing an n-dimensional hyperplane that optimally separates the data into two categories. (Sound familiar?) SVM models are closely related to neural networks. In fact, a particular SVM model can be shown to be equivalent to a two-layer, perceptron neural network. Consequently some of the ideas will be similar.

In this simplest form the SVM could be used to determine if an image was say a giraffe or a elephant. We would use a set of images of giraffes and elephants as our training data and then use our model to classify an image (which should be a giraffe or an elephant).

Consider again the case where we have a set of linearly separable data as illustrated in the figure below.

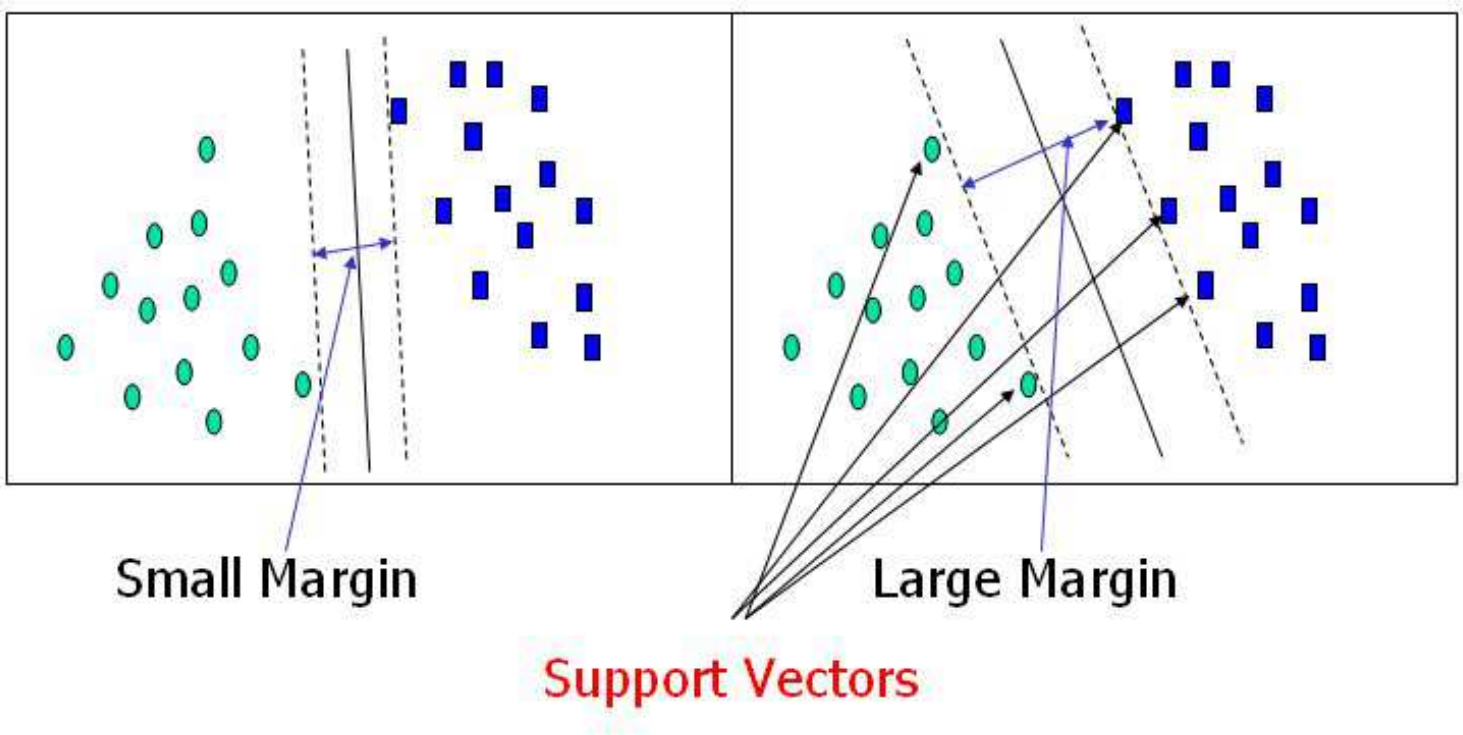


There are an infinite number of possible lines; two candidate lines are shown above. The question is which line is better, and how do we define the optimal line.

The dashed lines drawn parallel to the separating line mark the distance between the dividing line and the closest input data to the line.

The distance between the dashed lines is called the margin.

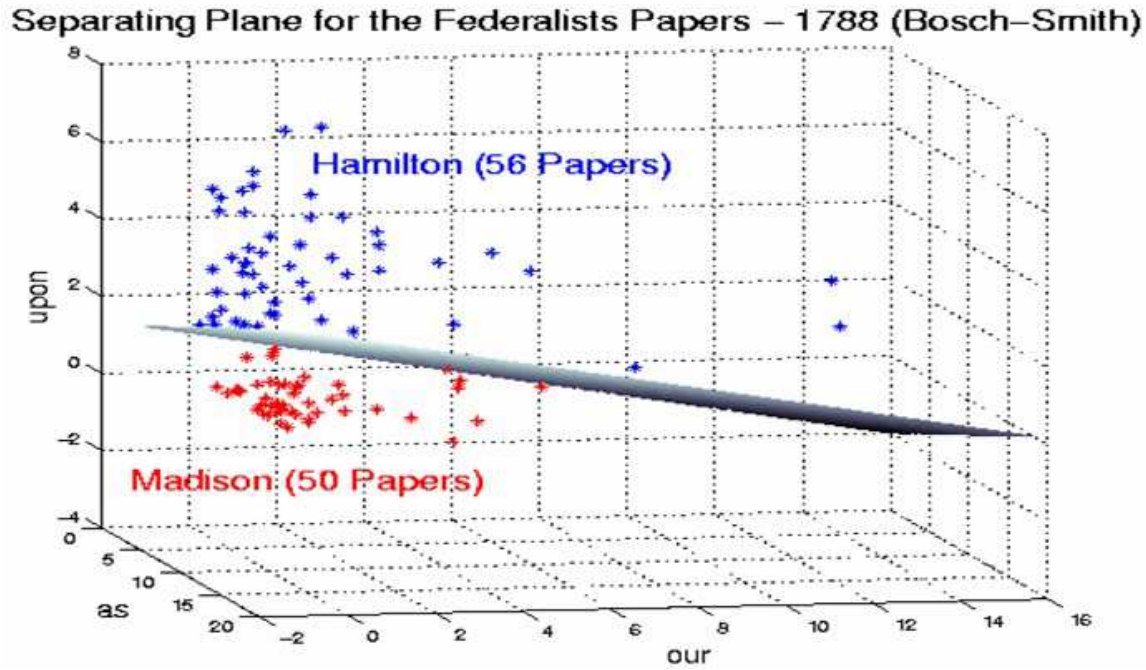
The input vectors (points) that constrain the width of the margin are the support vectors. The following figure illustrates this. Note that we just move our separating line to the left or the right until it touches the first data point.



Do we want a small margin or a large margin?

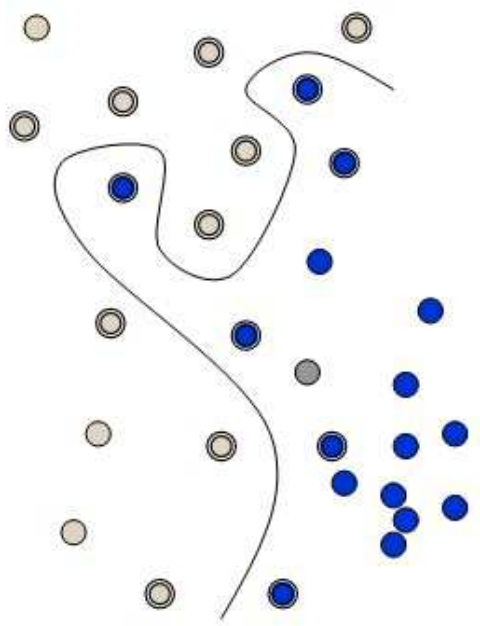
An SVM analysis finds the hyperplane (in this illustrated case, a line) that is oriented so that the margin between the support vectors is maximized. Even though the line found in both cases separates the training data there is no guarantee that the hyperplane will perform as well on new data. So intuitively it is to our advantage to find the line with the largest margin. In our previous figure the line in the right image is preferable to the line in the left image.

If all analyses consisted of two-category target variables with two predictor variables, and the cluster of points could be divided by a straight line, life would be easy. Unfortunately, this is not generally the case, so SVM must deal with (i) more than two predictor variables (i.e., input vectors in higher dimensions than 2), (ii) separating the points with non-linear curves, (iii) handling the cases where clusters cannot be completely separated, and (iv) handling classifications with more than two categories.



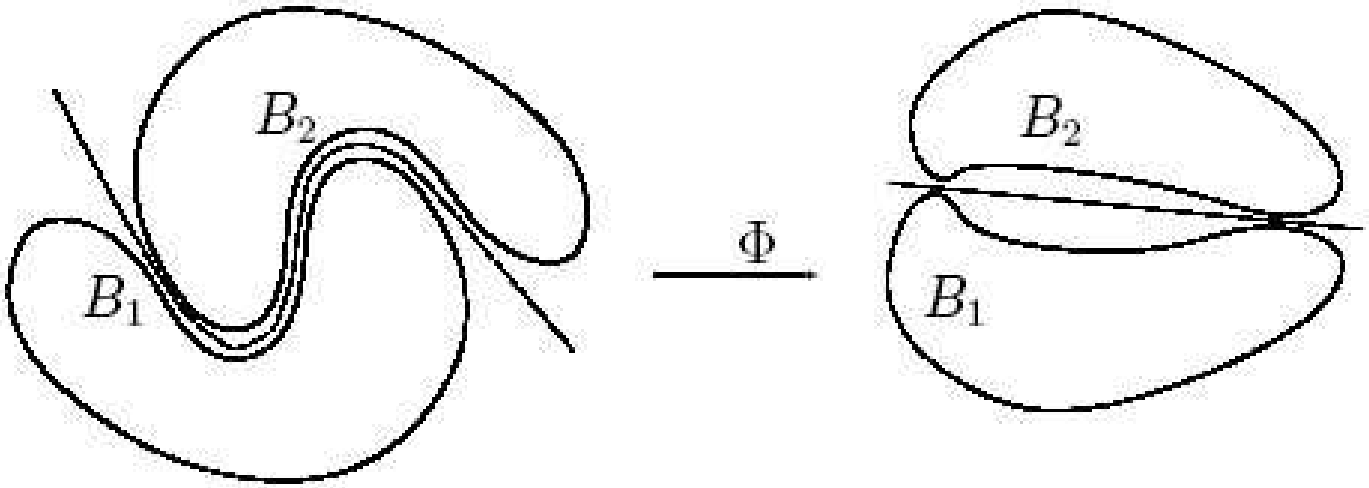
In the previous example our input data was vectors in $\mathbb{R}^2$. If we add a third variable, then we can use its value for a third dimension and plot the points in a 3-dimensional cube. Points on a 2-dimensional plane can be separated by a 1-dimensional line. Similarly, points in a 3-dimensional cube can be separated by a 2-dimensional plane.

The simplest way to divide two groups is with a straight line, a plane or an n-dimensional hyperplane. But what if the points are separated by a nonlinear region such as shown below?

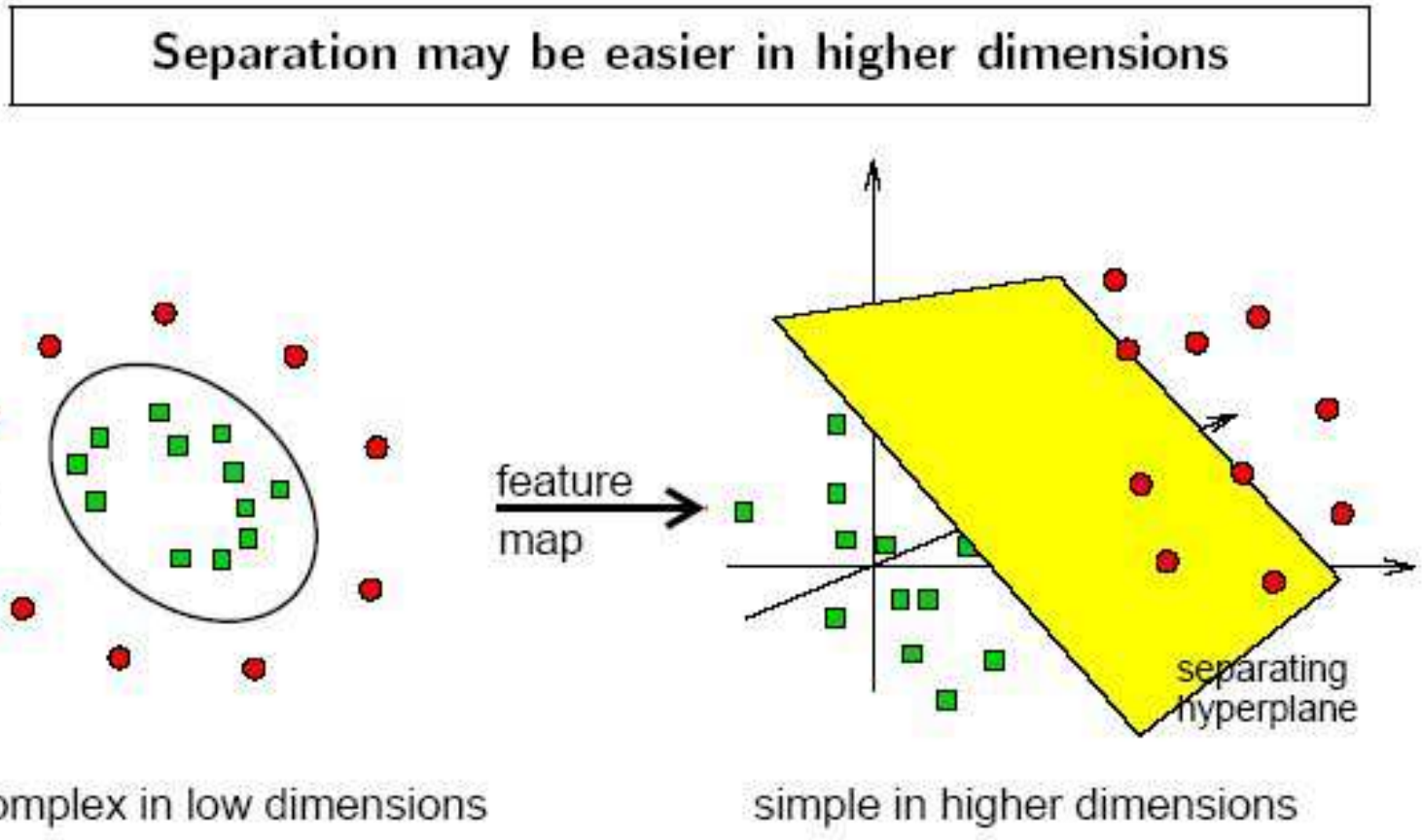


One option is to fit a curve through the data. SVM takes a different approach.

SVM handles this by using a kernel function to map the data into a different space where a hyperplane can be used to do the separation.



The kernel function may even transform the data into a higher dimensional space to make it possible to perform the separation.



Before we look at more complicated cases let's start with the linear separable case which we have looked at before. This is analogous to the perceptron case.

As before we want to find our decision boundary given by

$$\vec{w}^T\vec{x} + b = 0$$

except that now we want to choose it so the margin is maximized. So basically we need to get an equation for the margin and maximize it subject to a constraint that enforces separation of the data.

To see that the weight $\vec{w}$ is orthogonal to our decision line/hyperplane, we first take two vectors $\vec{x}_a$ and $\vec{x}_b$ that terminate on a line separating the data. The difference $\vec{x}_a - \vec{x}_b$ lies on the line and both satisfy our decision boundary equation so subtracting gives

$$\vec{w}^T(\vec{x}_a - \vec{x}_b) = 0$$

This says the weight $\vec{w}$ is orthogonal to our decision line.

As before we take

$$\vec{x} \in \mathcal{C}_1 \implies \vec{w}^T\vec{x} + b \geq 0$$

$$\vec{x} \in \mathcal{C}_2 \implies \vec{w}^T\vec{x} + b < 0$$

By convention we designate the class by $y = 1$ for $\mathcal{C}_1$ and $y = -1$ for $\mathcal{C}_2$. Doing this we can rewrite our conditions above in the compact form

$$y(\vec{w}^T\vec{x} + b) \geq 1$$

for data $(\vec{x}, y)$.

Now to get an equation for the margin we think of two lines which are parallel to our decision boundary. Assume the line to the right of the decision boundary first touches say $\vec{x}_1 \in \mathcal{C}_1$ and the one to the left touches $\vec{x}_2 \in \mathcal{C}_2$. Let d be the distance between these two lines.

We can rescale and get that

$$\vec{w}^T\vec{x}_1 + b = 1 \qquad \vec{w}^T\vec{x}_2 + b = -1$$

so that $\vec{w}^T(\vec{x}_1 - \vec{x}_2) = 2$ which says that the projection of $\vec{w}$ onto $(\vec{x}_1 - \vec{x}_2)$ has length 2 or equivalently that $\cos\theta\|\vec{w}\|_2\|\vec{x}_1 - \vec{x}_2\|_2 = 2$ where θ is the angle between the vectors. From trigonometry we have $\cos\theta = d/\|\vec{x}_1 - \vec{x}_2\|_2$ and combining gives

$$d = \frac{2}{\|w\|_2}$$

To cast this into a constrained minimization problem we first note that maximizing $d = \frac{2}{\|w\|_2}$ is equivalent to minimizing the reciprocal $\frac{\|w\|_2}{2}$. Also when we are dealing with the Euclidean norm it is usually easier to consider its square so we don't have to deal with square roots.

Given a set of data $(\vec{x}_i, y_i)$, $i = 1, 2, \ldots, m$ we seek $\vec{w}$ such that

$$\min_w \frac{\|w\|_2^2}{2}$$

subject to $y_i(\vec{w}^T \vec{x}_i + b) \geq 1$ for $i = 1, 2, \ldots, m$

This is a constrained minimization problem which can be solved by the standard technique of Lagrange multipliers.

Example Solve the problem

$\min f(x, y)$ where $f(x, y) = x + 2y$ subject to the constraint $x^2 + y^2 = 4$

using Lagrange multipliers.

We first introduce a Lagrange multiplier λ and write the Lagrangian

$$\mathcal{L}(x, y, \lambda) = x + 2y + \lambda(x^2 + y^2 - 4)$$

which we want to minimize. To do this, we simply takes the first partial derivatives and set to zero to get

$$\frac{\partial \mathcal{L}}{\partial x} = 1 + 2\lambda x \quad \frac{\partial \mathcal{L}}{\partial y} = 2 + 2\lambda y \quad \frac{\partial \mathcal{L}}{\partial \lambda} = x^2 + y^2 - 4$$

Note that $\frac{\partial \mathcal{L}}{\partial \lambda}$ gives our constraint. We now have the system of nonlinear equations

$$1 + 2\lambda x = 0 \quad 2 + 2\lambda y = 0 \quad x^2 + y^2 - 4$$

but they can be easily solved to give $\lambda = -\sqrt{5}/4$, $x = -2/\sqrt{5}$, $y = -4/\sqrt{5}$ because the nonlinear equation is just a quadratic.

In the case of equality constraints the Lagrange multipliers are free parameters and can take any values. Inequality constraints require a bit more work because the Lagrange multipliers are no longer free to take on any values.

For our problem we have m inequality constraints and we must constrain our Lagrange multipliers, i.e., they are not free to take on any values. After a bit of work, one can show that the equations become

$$\vec{w} = \sum_{i=1}^{m} \lambda_i y_i \vec{x}_i$$

$$\sum_{i=1}^{m} \lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

$$\lambda_i \Big[y_i(\vec{w}^T \vec{x}_i + b) - 1 \Big] = 0$$

At first glance this may seem intractable because we have a Lagrange multiplier for each record in the training data so if m is large we have a large nonlinear system. However, if we consider

$$\lambda_i \Big(y_i(\vec{w}^T \vec{x}_i + b) - 1 \Big) = 0$$

this says that $\lambda_i = 0$ unless the training data $(\vec{x}_i, y)$ satisfies $y_i(\vec{w}^T\vec{x}_i + b) = 1$ which means that $\vec{x}_i$ is a support vector; if the condition is not satisfied $\lambda = 0$ so this means that in reality, many of the $\lambda_i = 0$.

To efficiently solve this problem one converts it into one which only involves the Lagrange multipliers and the training data.

Now the problem is to extremize this function which in itself is not that easy. However, it can be done using quadratic programming (instead of linear programming) which you will look at in ACS II.

We now turn to the problem of classifying data where the decision boundary is no longer a hyperplane. Recall that the idea behind SVM for this problem is to transform the data using a kernel ϕ so that in the transformed space it can have a linear decision boundary.

The concept of the kernel is very powerful and is the basis of the success of the SVM as a classifier.

Many kernel mapping functions can be used but a few kernel functions have been found to work well in for a wide variety of applications. We can use linear or polynomial functions but the usually recommended kernel function is the Radial Basis Function (RBF).

Wikipedia definition: A radial basis function (RBF) ϕ is a real-valued function whose value depends only on the distance from the origin, so that $\phi(\|x\|)$; or alternatively on the distance from some other point c, called a center, so that $\phi(x, c) = \phi(\|x - c\|)$. Any function ϕ that satisfies the property is a radial function. The norm used is usually Euclidean distance, although other distance functions are possible. Sums of radial basis functions are typically used to ap-

proximate given functions. This approximation process can also be interpreted as a simple kind of neural network.

Clearly lots of functions are radial basis functions. Here are a couple of common ones.

1. Gaussian $\phi(r) = e^{-\beta r^2}$ for $\beta > 0$
2. multiquadric $\phi(\vec{x}) = \sqrt{1 + \|\vec{x}\|_2}$

Due to time constraints we will only look at an example using a polynomial kernel.

Consider the mapping $\phi : I\!R^2 \to I\!R^5$ defined by

$$\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

Recall that when we take the inner product $\vec{u}^T\vec{v}$ the larger it is, the more similar the vectors are. So let's take the inner product of two vectors $\vec{u} = (u_1, u_2), \vec{v} = (v_1, v_2) \in I\!R^2$ in the transformed space and see what happens.

$$\phi(\vec{u})^T\phi(\vec{v}) = u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 1 = (\vec{u}^T\vec{v} + 1)^2$$

We call $K(\vec{u}, \vec{v}) = \phi(\vec{u})^T\phi(\vec{v})$ our kernel function.

Separation may be easier in higher dimensions