该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Comp9417 Review

Lecture1 LinRegress

Aims

After a brief introduction to this course and the topics in it, this lecture will introduce you to machine learning approaches to the problem of numerical prediction. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- the supervised learning task of numeric prediction
- how linear regression solves the problem of numeric prediction
- fitting linear regression by least squares error criterion
- non-linear regression via linear-in-the-parameters models
- parameter estimation for regression
- local (nearest-neighbour) regression

Note: slides with titles marked * are for background only.

The most widely used categories of machine learning algorithms are:

- *Supervised learning* – output class (or label) is given
- *Unsupervised learning* – no output class is given

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

For the class of *symbolic* representations, machine learning is viewed as:

searching a space of **hypotheses** ...

represented in a formal hypothesis language (trees, rules, graphs ...).

For the class of *numeric* representations, machine learning is viewed as:

"searching" a space of **functions** ...

represented as mathematical models (linear equations, neural nets, ...).

We will look at the simplest model for numerical prediction:
a *regression equation*

Outcome will be a linear sum of feature values with appropriate weights.

Note: the term *regression* is overloaded – it can refer to:

- the process of determining the weights for the regression equation, or
- the regression equation itself.

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

- Numeric attributes and numeric prediction, i.e., regression
- Linear models, i.e. outcome is *linear* combination of attributes

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

- Weights are calculated from the training data
- **Predicted** value for first training instance $\mathbf{x}^{(1)}$ is:

$$b_0 x_0^{(1)} + b_1 x_1^{(1)} + b_2 x_2^{(1)} + \ldots + b_n x_n^{(1)} = \sum_{i=0}^{n} b_i x_i^{(1)}$$

$n+1$ coefficients are chosen so that sum of squared error on all instances in training data is minimized

Squared error:

$$\sum_{j=1}^{m} \left(y^{(j)} - \sum_{i=0}^{n} b_i x_i^{(j)} \right)^2$$

- For example, when a number of samples are drawn and the mean of each is found, then average of these means is equal to the population mean
- Such an estimator is said to be *statistically unbiased*

Mean. This is calculated as follows.

- Find the total T of N observations. Estimate the (arithmetic) mean by $m = T/N$.
- This works very well when the data follow a symmetric bell-shaped frequency distribution (of the kind modelled by "normal" distribution)
- A simple mathematical expression of this is $m = \frac{1}{N} \sum_i x_i$, where the observations are x_1, x_2 $\ldots$ x_n

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Variance. This is calculated as follows:

- Calculate the total T and the sum of squares of N observations. The estimate of the standard deviation is $s = \sqrt{\frac{1}{N-1}\sum_i (x_i - m)^2}$
- Again, this is a very good estimate when the data are modelled by a normal distribution

- Again, we have a similar formula in terms of expected values, for the scatter (spread) of values of a r.v. X around a mean value $E(X)$:

$$\begin{aligned} Var(X) &= E((X - E(X))^2) \\ &= E(X^2) - [E(X)]^2 \end{aligned}$$

- You can remember this as "the mean of the squares minus the square of the mean"

- The *correlation coefficient* is a number between -1 and +1 that indicates whether a pair of variables x and y are associated or not, and whether the scatter in the association is high or low
  - High values of x are associated with high values of y *and* low values of x are associated with low values of y, and scatter is low
  - A value near 0 indicates that there is no particular association and that there is a large scatter associated with the values
  - A value close to -1 suggests an inverse association between x and y

- The formula for computing correlation between x and y is:

$$r = \frac{\mathrm{cov}(x, y)}{\sqrt{\mathrm{var}(x)}\sqrt{\mathrm{var}(y)}}$$

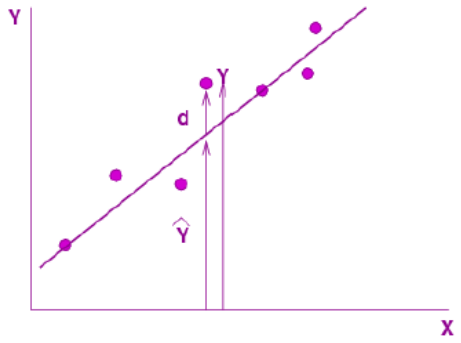
This is sometimes also called *Pearson's correlation coefficient*

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

- The terms in the denominator are simply the standard deviations of x and y. But the numerator is different. This is the *covariance*, calculated as the average of the product of deviations from the mean:

$$\text{cov}(x, y) = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{n - 1}$$

- MORAL: Do not use correlations to compare datasets. All you can derive is whether there is a positive or negative relationship between x and y



- GOAL: fit a line whose equation is of the form $\hat{Y} = a + bX$
- HOW: minimise $\sum_i d_i^2 = \sum_i (Y_i - \hat{Y}_i)^2$ (the "least squares estimator")

- The calculation for b is given by:

$$b = \frac{\text{cov}(x, y)}{\text{var}(x)}$$

where $\text{cov}(x, y)$ is the covariance of x and y, given by $\sum_i (x_i - \overline{x})(y_i - \overline{y})$ as before

- $a = \overline{Y} - b\overline{X}$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Another important point to note is that the sum of the residuals of the least-squares solution is zero:

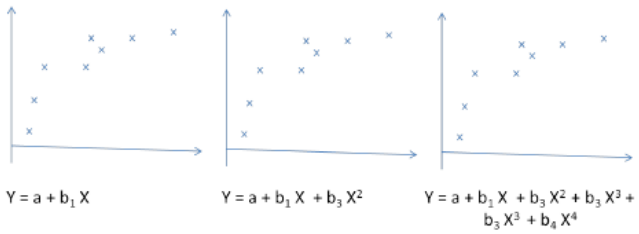
$$\sum_{i=1}^{n}(y_i - (\hat{a} + \hat{b}x_i)) = n(\overline{y} - \hat{a} - \hat{b}\overline{x}) = 0$$

The result follows because $\hat{a} = \overline{Y} - \hat{b}\overline{X}$, as derived above.

Regularisation is a general method to avoid overfitting by applying additional constraints to the weight vector. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*.

Recall the setting for regression in terms of cost minimization.

- Can add penalty terms to a *cost* function, forcing coefficients to shrink to zero



$$Y = f_{\theta_0,\theta_1,\ldots,\theta_n}(X_1, X_2, \ldots, X_n) = f_\theta(\mathbf{X})$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

- MSE as a cost function, given data $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)$

$$Cost(\theta) = \frac{1}{n}\sum_i (f_\theta(\mathbf{x_i}) - y_i)^2$$

and with a penalty function:

$$Cost(\theta) = \frac{1}{n}\sum_i (f_\theta(\mathbf{x_i}) - y_i)^2 + \frac{1}{n}\lambda\sum_{i=1}^{n}\theta_i$$

- Parameter estimation by optimisation will attempt to values for $\theta_0, \theta_1, \ldots, \theta_n$ s.t. $Cost(\theta)$ is a minimum
- It will be easier to take the $\frac{1}{n}$ term as $\frac{1}{2n}$, which will not affect the minimisation

The multivariate least-squares regression problem can be written as an optimisation problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

The regularised version of this optimisation is then as follows:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda||\mathbf{w}||^2$$

where $||\mathbf{w}||^2 = \sum_i w_i^2$ is the squared norm of the vector $\mathbf{w}$, or, equivalently, the dot product $\mathbf{w}^{\mathrm{T}}\mathbf{w}$; λ is a scalar determining the amount of regularisation.

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

This regularised problem still has a closed-form solution:

$$\hat{\mathbf{w}} \quad = \quad (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

where $\mathbf{I}$ denotes the identity matrix. Regularisation amounts to adding λ to the diagonal of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-squares regression is known as *ridge regression*.

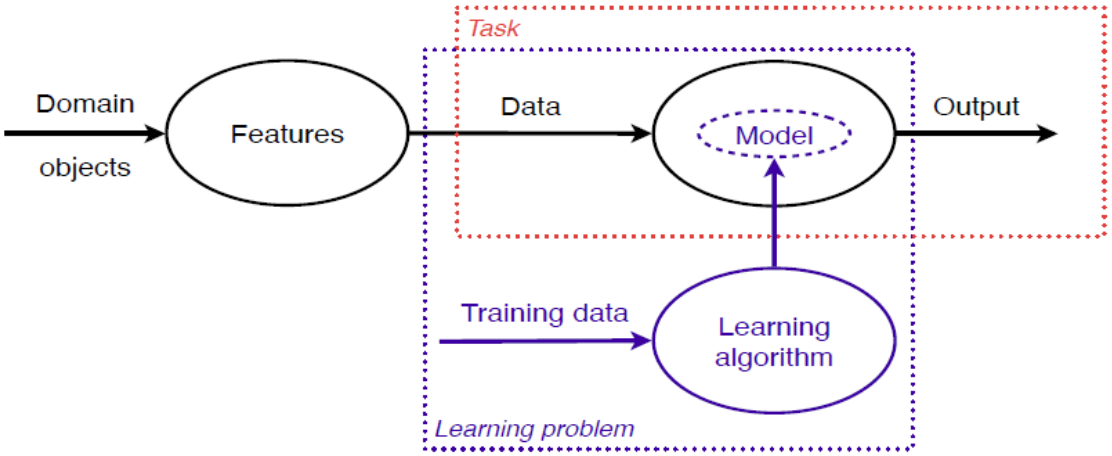
该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Lecture2 Classification1

Aims

This lecture will introduce you to machine learning approaches to the problem of *classification*. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- outline a framework for solving machine learning problems
- outline the general problem of induction
- describe issues of generalisation and evaluation for classification
- outline the use of a linear model as a 2-class classifier
- describe distance measures and how they are used in classification
- outline the basic k-nearest neighbour classification method



该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Deduction: derive specific consequences from general theories

Induction: derive general theories from specific observations

Deduction is well-founded (mathematical logic).

Induction is (philosophically) problematic – induction is useful since it often seems to work – an inductive argument !

The inductive learning hypothesis

> *Any hypothesis found to approximate the target (true) function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.*

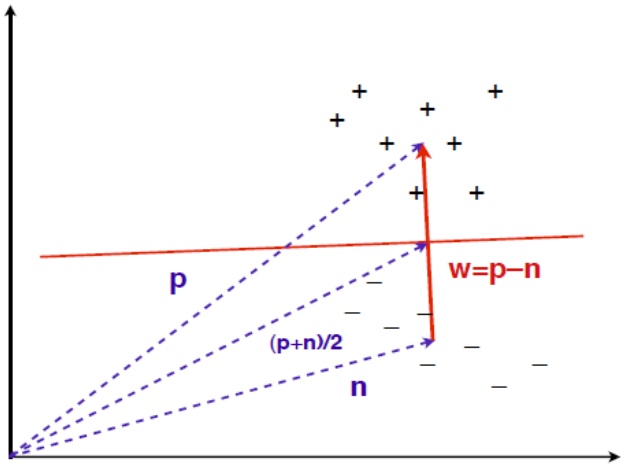
Does the algorithm require all training data to be present before the start of learning ? If yes, then it is categorised as **batch learning** algorithm.

If however, it can continue to learn a new data arrives, it is an **online learning** algorithm.

If the model has a fixed number of parameters it is categorised as **parametric**.

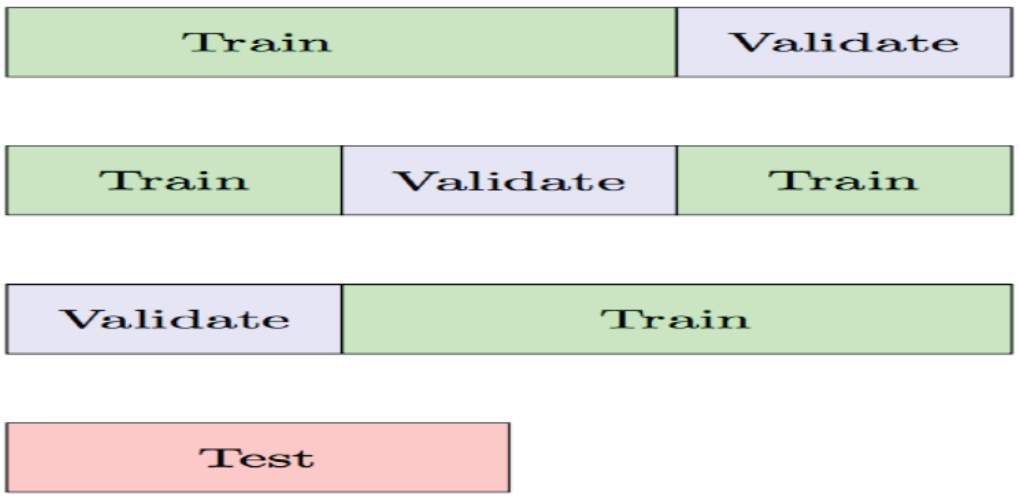
Otherwise, if the number of parameters grows with the amount of training data it is categorised as **non-parametric**.

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/



The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass.

The basic linear classifier is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting that $(\mathbf{p} + \mathbf{n})/2$ is on the decision boundary, and hence $t = (\mathbf{p} - \mathbf{n}) \cdot (\mathbf{p} + \mathbf{n})/2 = (||\mathbf{p}||^2 - ||\mathbf{n}||^2)/2$, where $||\mathbf{x}||$ denotes the length of vector $\mathbf{x}$.



该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Two-class prediction case:

Actual Class	Predicted Class	
	Yes	No
Yes	True Positive (TP)	False Negative (FN)
No	False Positive (FP)	True Negative (TN)

$$\text{acc} = \frac{1}{|\text{Test}|} \sum_{x \in \text{Test}} I[\hat{c}(x) = c(x)]$$

Minkowski distance If $\mathcal{X} = \mathbb{R}^d$, the *Minkowski distance* of order $p > 0$ is defined as

$$\text{Dis}_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^{d} |x_j - y_j|^p \right)^{1/p} = ||\mathbf{x} - \mathbf{y}||_p$$

where $||\mathbf{z}||_p = \left(\sum_{j=1}^{d} |z_j|^p \right)^{1/p}$ is the p-*norm* (sometimes denoted L_p norm) of the vector $\mathbf{z}$.

Hamming distance

$$\text{Dis}_0(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{d} (x_j - y_j)^0 = \sum_{j=1}^{d} I[x_j = y_j]$$

- The 1-*norm* denotes *Manhattan distance*, also called *cityblock distance*:

$$\text{Dis}_1(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{d} |x_j - y_j|$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

- The 2-norm refers to the familiar *Euclidean distance*

$$\text{Dis}_2(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{j=1}^{d}(x_j - y_j)^2} = \sqrt{(\mathbf{x}-\mathbf{y})^{\mathrm{T}}(\mathbf{x}-\mathbf{y})}$$

- If we now let p grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that $\text{Dis}_\infty(\mathbf{x},\mathbf{y}) = \max_j |x_j - y_j|$; this is also called *Chebyshev distance*.

Distance metric Given an instance space $\mathcal{X}$, a *distance metric* is a function $\text{Dis} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any $x, y, z \in \mathcal{X}$:

- distances between a point and itself are zero: $\text{Dis}(x,x) = 0$;
- all other distances are larger than zero: if $x \neq y$ then $\text{Dis}(x,y) > 0$;
- distances are symmetric: $\text{Dis}(y,x) = \text{Dis}(x,y)$;
- detours can not shorten the distance: $\text{Dis}(x,z) \leq \text{Dis}(x,y) + \text{Dis}(y,z)$.

If the second condition is weakened to a non-strict inequality – i.e., $\text{Dis}(x,y)$ may be zero even if $x \neq y$ – the function Dis is called a *pseudo-metric*.

Stores all training examples $\langle x_i, f(x_i)\rangle$.

Nearest neighbour:

- Given query instance x_q, first locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

k-Nearest neighbour:

- Given x_q, take vote among its k nearest neighbours (if discrete-valued target function)
- take mean of f values of k nearest neighbours (if real-valued)

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Training algorithm:

- For each training example $\langle x_i, f(x_i)\rangle$, add the example to the list *training_examples*.

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ be the k instances from *training_examples* that are *nearest* to x_q by the distance function
  - Return

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

    where $\delta(a, b) = 1$ if $a = b$ and 0 otherwise.

- Different attributes measured on different scales
- Need to be *normalized* (why ?)

$$a_r = \frac{v_r - \min v_r}{\max v_r - -\min v_r}$$

  where v_r is the actual value of attribute r

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

Advantages:

- Statisticians have used k-NN since early 1950s
- Can be very accurate
  - at most twice the "Bayes error" for 1-NN (Cover & Hart, 1967)
- Training is very fast
- Can learn complex target functions
- Don't lose information by generalization - keep all instances

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Disadvantages:

- Slow at query time: basic algorithm scans entire training data to derive a prediction
- "Curse of dimensionality"
- Assumes all attributes are equally important, so easily fooled by irrelevant attributes
  - Remedy: attribute selection or weights
- Problem of noisy instances:
  - Remedy: remove from data set
  - not easy – how to know which are noisy ?

What is the inductive bias of k-NN ?

- an assumption that the classification of query instance x_q will be most similar to the classification of other instances that are nearby according to the distance function
- kNN uses the training data as exemplars, so training is $O(n)$ (but prediction is also $O(n)$!)
- 1NN perfectly separates training data, so low bias but high variance
- By increasing the number of neighbours k we increase bias and decrease variance (what happens when $k = n$?)
- Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when $k > 1$
- Warning: in high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality)

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

- Might want to weight nearer neighbours more heavily ...
- Use distance function to construct a weight w_i
- Replace the final line of the classification algorithm by:

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between x_q and x_i

For real-valued target functions replace the final line of the algorithm by:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

(denominator normalizes contribution of individual weights).
Now we can consider using *all* the training examples instead of just k

$\rightarrow$ using all examples (i.e., when $k = n$) with the rule above is called Shepard's method

Lazy learners do not construct an explicit model, so how do we evaluate the output of the learning process ?

- 1-NN – training set error is always zero !
  - each training example is always closest to itself
- k-NN – overfitting may be hard to detect

Leave-one-out cross-validation (LOOCV) – leave out each example and predict it given the rest:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)$$

Error is mean over all predicted examples. Fast – no models to be built !

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Recap – Practical problems of 1-NN scheme:

- Slow (but fast $k-d$ tree-based approaches exist)
  - Remedy: removing irrelevant data
- Noise (but k-NN copes quite well with noise)
  - Remedy: removing noisy instances
- All attributes deemed equally important
  - Remedy: attribute weighting (or simply selection)
- Doesn't perform explicit generalization
  - Remedy: rule-based or tree-based NN approaches

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Lecture3 Classifiaction2

Aims

This lecture will continue your exposure to machine learning approaches to the problem of *classification*. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- explain the concept of inductive bias in machine learning
- outline Bayes Theorem as applied in machine learning
- define MAP and ML inference using Bayes theorem
- define the Bayes optimal classification rule in terms of MAP inference
- outline the Naive Bayes classification algorithm
- describe typical applications of Naive Bayes for text classification
- outline the logistic regression classification algorithm

Confusingly, "inductive bias" is *NOT* the same "bias" as in the "bias-variance" decomposition.

"Inductive bias" is the combination of assumptions and restrictions placed on the models and algorithms used to solve a learning problem.

Essentially it means that the algorithm and model combination you are using to solve the learning problem is appropriate for the task.

Success in machine learning requires understanding the inductive bias of algorithms and models, and choosing them appropriately for the task.

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

For example, what is the inductive bias of:

- Linear Regression ?
  - target function has the form $y = ax + b$
  - approximate by fitting using MSE
- Nearest Neighbour ?
  - target function is a complex non-linear function of the data
  - predict using nearest neighbour by Euclidean distance in feature space

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

where

$P(h)$ = 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

Maximum a posteriori hypothesis h_{MAP}:

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h|D) \\ &= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg\max_{h \in H} P(D|h)P(h) \end{aligned}$$

Maximum likelihood (ML) hypothesis

$$h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Product Rule: probability $P(A \wedge B)$ of conjunction of two events A and B:

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

Sum Rule: probability of disjunction of two events A and B:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Theorem of total probability: if events $A_1, \ldots, A_n$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$, then:

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

- *Conditional Probability*: probability of A given B:

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

- Rearrange sum rule to get:

$$P(A \wedge B) = P(A) + P(B) - P(A \vee B)$$

Canonical concept learning task:

- instance space X, hypothesis space H, training examples D
- consider a learning algorithm that outputs most specific hypothesis from the *version space* $VS_{H,D}$ (i.e., set of all consistent or "zero-error" classification rules)

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Brute Force MAP Framework for Concept Learning:

Assume fixed set of instances $\langle x_1, \ldots, x_m \rangle$

Assume D is the set of classifications $D = \langle c(x_1), \ldots, c(x_m) \rangle$

Choose $P(h)$ to be *uniform* distribution:

- $P(h) = \frac{1}{|H|}$ for all h in H

Choose $P(D|h)$:

- $P(D|h) = 1$ if h consistent with D
- $P(D|h) = 0$ otherwise

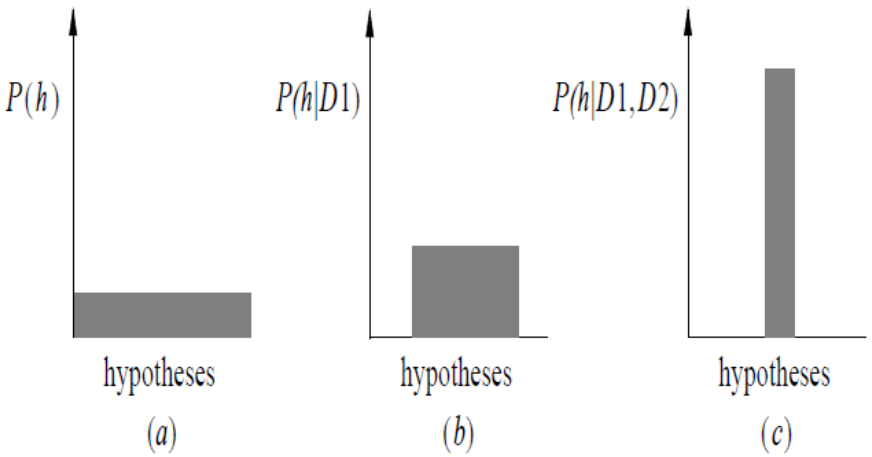
$$\begin{aligned} P(D) &= \sum_{h_i \in H} P(D|H_i)P(h_i) \\ &= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|} \\ &= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} \\ &= \frac{|VS_{H,D}|}{|H|} \end{aligned}$$

$$\begin{aligned} P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H,D}|} \end{aligned}$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Then:

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$



Consider any 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 some Gaussian (normal) distribution with mean=0

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

$$h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

$$
\begin{aligned}
h_{ML} &= \arg\max_{h \in H} p(D|h) \\
&= \arg\max_{h \in H} \prod_{i=1}^{m} p(d_i|h) \\
&= \arg\max_{h \in H} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{d_i - h(x_i)}{\sigma})^2}
\end{aligned}
$$

Recall that we treat each probability $p(D|h)$ **as if** $h = f$, i.e., **we assume** $\mu = f(x_i) = h(x_i)$, which is the key idea behind maximum likelihood !

Maximize natural log to give simpler expression:

$$
\begin{aligned}
h_{ML} &= \arg\max_{h \in H} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2} \left(\frac{d_i - h(x_i)}{\sigma} \right)^2 \\
&= \arg\max_{h \in H} \sum_{i=1}^{m} -\frac{1}{2} \left(\frac{d_i - h(x_i)}{\sigma} \right)^2 \\
&= \arg\max_{h \in H} \sum_{i=1}^{m} -(d_i - h(x_i))^2
\end{aligned}
$$

Equivalently, we can minimize the positive version of the expression:

$$
h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2
$$

- Three possible hypotheses:
  $P(h_1|D) = .4,\ P(h_2|D) = .3,\ P(h_3|D) = .3$
- Given new instance x,
  $h_1(x) = +,\ h_2(x) = -,\ h_3(x) = -$
- What's most probable classification of x?

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Bayes optimal classification:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Example:

$$\begin{aligned} P(h_1|D) &= .4, \quad P(-|h_1) = 0, \quad P(+|h_1) = 1 \\ P(h_2|D) &= .3, \quad P(-|h_2) = 1, \quad P(+|h_2) = 0 \\ P(h_3|D) &= .3, \quad P(-|h_3) = 1, \quad P(+|h_3) = 0 \end{aligned}$$

therefore

$$\begin{aligned} \sum_{h_i \in H} P(+|h_i)P(h_i|D) &= .4 \\ \sum_{h_i \in H} P(-|h_i)P(h_i|D) &= .6 \end{aligned}$$

and

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Assume target function $f : X \to V$, where each instance x described by attributes $\langle a_1, a_2 \ldots a_n \rangle$.
Most probable value of $f(x)$ is:

$$\begin{aligned} v_{MAP} &= \arg\max_{v_j \in V} P(v_j|a_1, a_2 \ldots a_n) \\ v_{MAP} &= \arg\max_{v_j \in V} \frac{P(a_1, a_2 \ldots a_n|v_j)P(v_j)}{P(a_1, a_2 \ldots a_n)} \\ &= \arg\max_{v_j \in V} P(a_1, a_2 \ldots a_n|v_j)P(v_j) \end{aligned}$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Naive Bayes assumption:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

- Attributes are statistically independent (given the class value)
  - which means knowledge about the value of a particular attribute tells us nothing about the value of another attribute (if the class is known)

which gives

Naive Bayes classifier: $v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$

Naive_Bayes_Learn($examples$)

For each target value v_j

$\hat{P}(v_j) \leftarrow$ estimate $P(v_j)$

For each attribute value a_i of each attribute a

$\hat{P}(a_i | v_j) \leftarrow$ estimate $P(a_i | v_j)$

Classify_New_Instance(x)

$$v_{NB} = \arg\max_{v_j \in V} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i | v_j)$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

What are the required probabilities to predict *PlayTennis* ?

Outlook			Temperature			Humidity			Windy		
	Yes	No		Yes	No		Yes	No		Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3
Rainy	3	2	Cool	3	1						
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5
Rainy	3/9	2/5	Cool	3/9	1/5						

Play	
Yes	No
9	5
9/14	5/14

Say we have the new instance:

$$\langle Outlk = sun, Temp = cool, Humid = high, Wind = true \rangle$$

We want to compute:

$$v_{NB} = \underset{v_j \in \{\text{“yes”}, \text{“no”}\}}{\arg\max} P(v_j) \prod_i P(a_i | v_j)$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

So we first calculate the likelihood of the two classes, "yes" and "no"

$$\begin{aligned} \text{for "yes"} &= P(y) \times P(sun|y) \times P(cool|y) \times P(high|y) \times P(true|y) \\ 0.0053 &= \frac{9}{14} \times \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \end{aligned}$$

$$\begin{aligned} \text{for "no"} &= P(n) \times P(sun|n) \times P(cool|n) \times P(high|n) \times P(true|n) \\ 0.0206 &= \frac{5}{14} \times \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \end{aligned}$$

Then convert to a probability by normalisation

$$\begin{aligned} P(\text{"yes"}) &= \frac{0.0053}{(0.0053 + 0.0206)} \\ &= 0.205 \\ P(\text{"no"}) &= \frac{0.0206}{(0.0053 + 0.0206)} \\ &= 0.795 \end{aligned}$$

The Naive Bayes classification is "no".

- Weights don't need to be equal (if they sum to 1) – a form of *prior*

Sunny	*Overcast*	*Rainy*
$\frac{2+\mu p_1}{9+\mu}$	$\frac{4+\mu p_2}{9+\mu}$	$\frac{3+\mu p_3}{9+\mu}$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Consider the following e-mails consisting of five words a, b, c, d, e:

e_1: $b\ d\ e\ b\ b\ d\ e$
e_2: $b\ c\ e\ b\ b\ d\ d\ e\ c\ c$
e_3: $a\ d\ a\ d\ e\ a\ e\ e$
e_4: $b\ a\ d\ b\ e\ d\ a\ b$
e_5: $a\ b\ a\ b\ a\ b\ a\ e\ d$
e_6: $a\ c\ a\ c\ a\ c\ a\ e\ d$
e_7: $e\ a\ e\ d\ a\ e\ a$
e_8: $d\ e\ d\ e\ d$

We are told that the e-mails on the left are spam and those on the right are ham, and so we use them as a small training set to train our Bayesian classifier.

E-mail	$a?$	$b?$	$c?$	Class
e_1	0	1	0	$+$
e_2	0	1	1	$+$
e_3	1	0	0	$+$
e_4	1	1	0	$+$
e_5	1	1	0	$-$
e_6	1	0	1	$-$
e_7	1	0	0	$-$
e_8	0	0	0	$-$

E-mail	$\#a$	$\#b$	$\#c$	Class
e_1	0	3	0	$+$
e_2	0	3	3	$+$
e_3	3	0	0	$+$
e_4	2	3	0	$+$
e_5	4	3	0	$-$
e_6	4	0	3	$-$
e_7	3	0	0	$-$
e_8	0	0	0	$-$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

In the case of a two-class problem, model the probability of one class $P(Y = 1)$ vs. the alternative $(1 - P(Y = 1))$:

$$P(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

or

$$\ln\frac{P(Y = 1|\mathbf{x})}{1 - P(Y = 1|\mathbf{x})} = \mathbf{w}^T\mathbf{x}$$

The quantity on the l.h.s. is called the *logit* and all we are doing is a linear model for the logit.
Note: to fit this is actually more complex than linear regression, so we omit the details.
Generalises to multiple class versions (Y can have more than two values).

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Lecture4 Classifiaction2

Aims

This lecture will enable you to describe decision tree learning, the use of entropy and the problem of overfitting. Following it you should be able to:

- define the decision tree representation
- list representation properties of data and models for which decision trees are appropriate
- reproduce the basic top-down algorithm for decision tree induction (TDIDT)
- define entropy in the context of learning a Boolean classifier from examples
- describe the inductive bias of the basic TDIDT algorithm
- define overfitting of a training set by a hypothesis
- describe developments of the basic TDIDT algorithm: pruning, rule generation, numerical attributes, many-valued attributes, costs, missing values
- describe regression and model trees

When to Consider Decision Trees?

- Instances described by a mix of numeric features and discrete attribute–value pairs
- Target function is discrete valued (otherwise use regression trees)
- Disjunctive hypothesis may be required
- Possibly noisy training data
- Interpretability is an advantage

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Main loop:

- $A \leftarrow$ the "best" decision attribute for next $node$
- Assign A as decision attribute for $node$
- For each value of A, create new descendant of $node$
- Sort training examples to leaf nodes
- If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Suppose X can have one of m values $\ldots V_1, V_2, \ldots V_m$

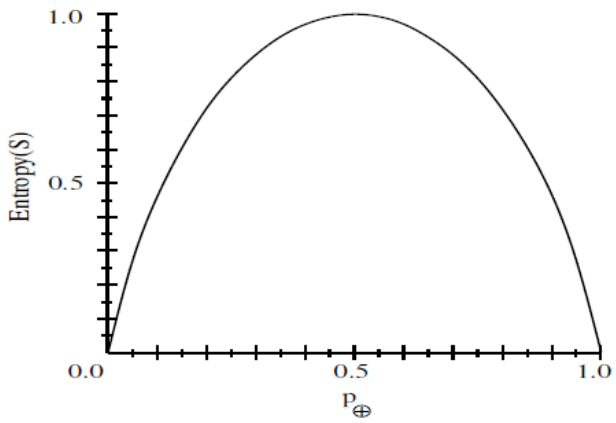
$P(X = V_1) = p_1$	$P(X = V_2) = p_2$	$\ldots$	$P(X = V_m) = p_m$

What's the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X's distribution ? It's

$$\begin{aligned} H(X) &= -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_m \log_2 p_m \\ &= -\sum_{j=1}^{m} p_j \log_2 p_j \end{aligned}$$

$H(X) =$ the *entropy* of X

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/



Where:
S is a sample of training examples
$p_{\oplus}$ is the proportion of positive examples in S
$p_{\ominus}$ is the proportion of negative examples in S

- $Gain(S, A)$ = expected reduction in entropy due to sorting on A

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Inductive Bias in ID3

Note H is the power set of instances X

→Unbiased?

Not really...

- Preference for short trees, and for those with high information gain attributes near the root
- Bias is a *preference* for some hypotheses, rather than a *restriction* of hypothesis space H
- an incomplete search of a complete hypothesis space *versus* a complete search of an incomplete hypothesis space (as in learning conjunctive concepts)
- Occam's razor: prefer the shortest hypothesis that fits the data

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

Definition

Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

How can we avoid overfitting? **Pruning**

- **pre-pruning** stop growing when data split not statistically significant
- **post-pruning** grow full tree, then remove sub-trees which are overfitting

- Pre-pruning may suffer from early stopping: may stop the growth of tree prematurely

- And: pre-pruning faster than post-pruning

Post-pruning

- Builds full tree first and prunes it afterwards
  - Attribute interactions are visible in fully-grown tree
- Problem: identification of subtrees and nodes that are due to chance effects
- Two main pruning operations:
  - Subtree replacement
  - Subtree raising
- Possible strategies: error estimation, significance testing, MDL principle
- We examine two methods: Reduced-error Pruning and Error-based Pruning

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Reduced-Error Pruning

- **Good** produces smallest version of most accurate subtree
- **Not so good** reduces effective size of training set

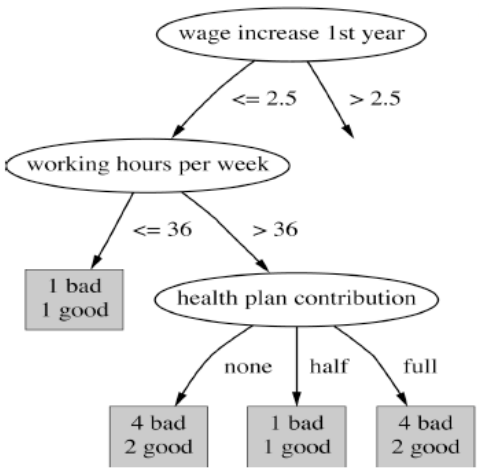
该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Error-based pruning: error estimate

- The error estimate for a tree node is the weighted sum of error estimates for all its subtrees (possibly leaves).
- Upper bound error estimate e for a node (simplified version):

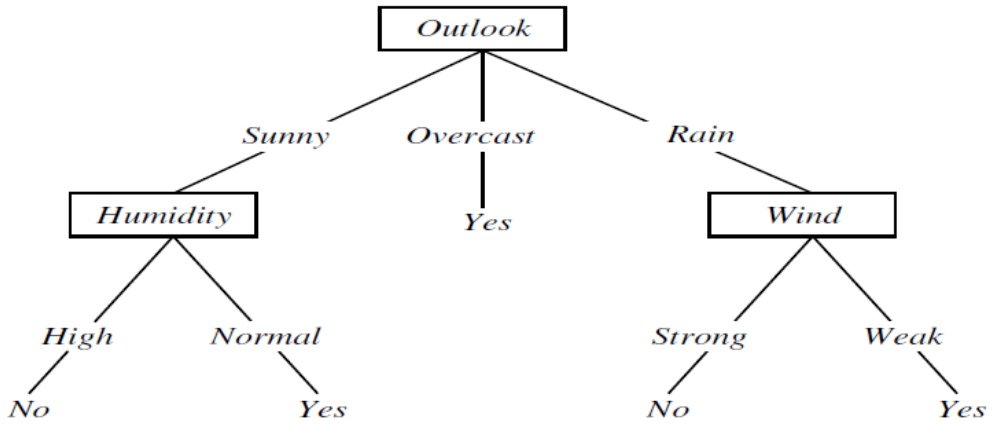
$$e = f + Z_c \cdot \sqrt{\frac{f \cdot (1 - f)}{N}}$$

- f is actual (empirical) error of tree on examples at the tree node
- N is the number of examples at the tree node
- Z_c is a constant whose value depends on *confidence* parameter c
- C4.5's default value for confidence $c = 0.25$
- If $c = 0.25$ then $Z_c = 0.69$ (from standardized normal distribution)



- health plan contribution: node measures $f = 0.36$, $e = 0.46$
- sub-tree measures:
  - none: $f = 0.33$, $e = 0.47$
  - half: $f = 0.5$, $e = 0.72$
  - full: $f = 0.33$, $e = 0.47$
- sub-trees combined $6 : 2 : 6$ gives 0.51
- sub-trees estimated to give *greater* error so prune away

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/



IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

IF $(Outlook = Sunny) \wedge (Humidity = Normal)$
THEN $PlayTennis = Yes$

Problem:

- If attribute has many values, $Gain$ will select it
- Why ? more likely to split instances into "pure" subsets
  - Maximised by singleton subsets
- Imagine using *Date* = June 4, 2019 as attribute
- High gain on training set, useless for prediction

One approach: use $GainRatio$ instead

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where S_i is subset of S for which A has value v_i

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Consider

- medical diagnosis, $BloodTest$ has cost \$150
- robotics, $Width\_from\_1ft$ has cost 23 sec.

How to learn a consistent tree with low expected cost?

One approach: evaluate information gain *relative to* cost:

- Example

$$\frac{Gain^2(S, A)}{Cost(A)}.$$

Preference for decision trees using lower-cost attributes.

Windowing

Early implementations – training sets too large for memory

As a solution ID3 implemented *windowing*:

1. select subset of instances – the *window*
2. construct decision tree from all instances in the window
3. use tree to classify training instances *not* in window
4. if all instances correctly classified then halt, else
5. add selected misclassified instances to the window
6. go to step 2

Windowing retained in C4.5 because it can lead to *more accurate* trees.
Related to *ensemble learning*.

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Regression trees

- Differences to decision trees:
  - Splitting criterion: minimizing intra-subset variation
  - Pruning criterion: based on numeric error measure
  - Leaf node predicts average class values of training instances reaching that node
- Can approximate piecewise constant functions
- Easy to interpret
- More sophisticated version: model trees

- In regression problems we can define the variance in the usual way:

$$\mathrm{Var}(Y) = \frac{1}{|Y|}\sum_{y\in Y}(y-\overline{y})^2$$

If a split partitions the set of target values Y into mutually exclusive sets $\{Y_1,\ldots,Y_l\}$, the weighted average variance is then

$$\mathrm{Var}(\{Y_1,\ldots,Y_l\}) = \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\mathrm{Var}(Y_j) = \ldots = \frac{1}{|Y|}\sum_{y\in Y}y^2 - \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\overline{y}_j^2$$

The first term is constant for a given set Y and so we want to maximise the weighted average of squared means in the children.

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:

#	Model	Condition	Leslie	Price
1.	B3	excellent	no	4513
2.	T202	fair	yes	625
3.	A100	good	no	1051
4.	T202	good	no	270
5.	M102	good	yes	870
6.	A100	excellent	no	1770
7.	T202	fair	no	99
8.	A100	good	yes	1900
9.	E112	fair	no	77

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase.
There are three features, hence three possible splits:

Model = [A100, B3, E112, M102, T202]
$[1051, 1770, 1900][4513][77][870][99, 270, 625]$
Condition = [excellent, good, fair]
$[1770, 4513][270, 870, 1051, 1900][77, 99, 625]$
Leslie = [yes, no] $[625, 870, 1900][77, 99, 270, 1051, 1770, 4513]$

The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of squared means is $3.21 \cdot 10^6$. The means of the second split are 3142, 1023 and 267, with weighted average of squared means $2.68 \cdot 10^6$; for the third split the means are 1132 and 1297, with weighted average of squared means $1.55 \cdot 10^6$. We therefore branch on Model at the top level. This gives us three single-instance leaves, as well as three A100s and three T202s.

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

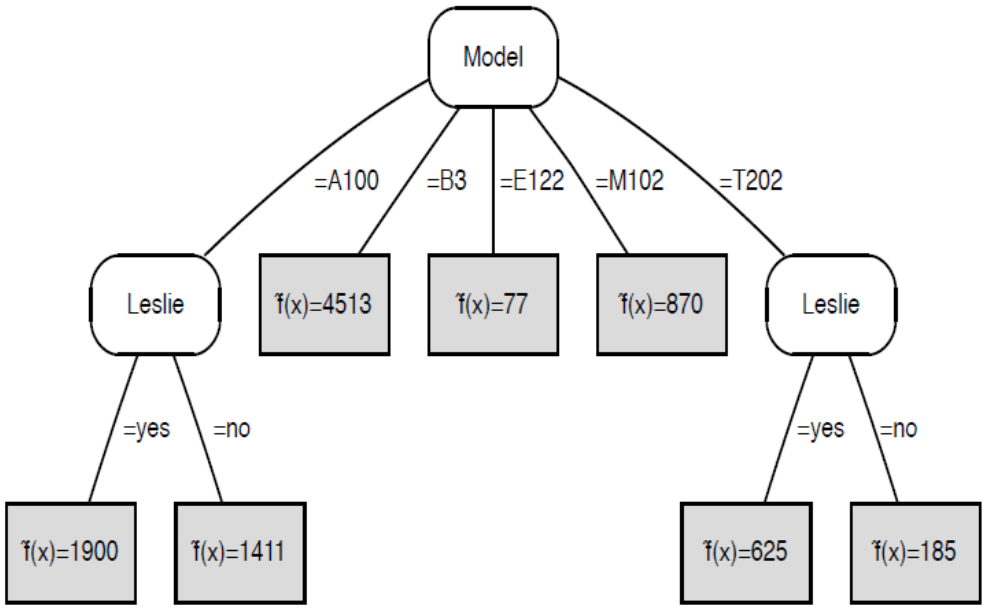
For the A100s we obtain the following splits:

Condition = [excellent, good, fair] $[1770][1051, 1900][]$
Leslie = [yes, no] $[1900][1051, 1770]$

Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

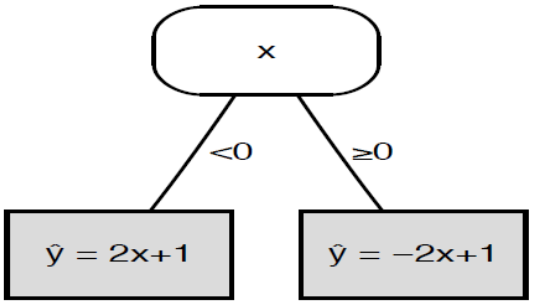
Condition = [excellent, good, fair] $[][270][99, 625]$
Leslie = [yes, no] $[625][99, 270]$

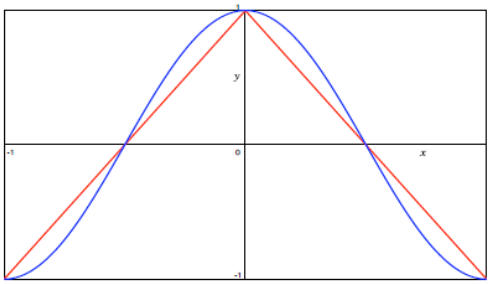
Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted on the next slide.



该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

A small model tree





Smoothing

- Naïve prediction method – output value of LR model at corresponding leaf node
- Improve performance by *smoothing* predictions with *internal* LR models
  - Predicted value is weighted average of LR models along path from root to leaf
- Smoothing formula: $p' = \frac{np+kq}{n+k}$ where
  - p' prediction passed up to next higher node
  - p prediction passed to this node from below
  - q value predicted by model at this node
  - n number of instances that reach node below
  - k smoothing constant
- Same effect can be achieved by incorporating the internal models into the leaf nodes

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Building the tree

- Splitting criterion: *standard deviation reduction*

$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

where $T_1, T_2, \ldots$ are the sets from splits of data at node.

- Termination criteria (important when building trees for numeric prediction):
  - Standard deviation becomes smaller than certain fraction of sd for full training set (e.g. 5%)
  - Too few instances remain (e.g. less than four)

Pruning the tree

- Pruning is based on estimated absolute error of LR models
- Heuristic estimate:

$$\frac{n+v}{n-v} \times \text{average\_absolute\_error}$$

where n is number of training instances that reach the node, and v is the number of parameters in the linear model

- LR models are pruned by greedily removing terms to minimize the estimated error
- Model trees allow for heavy pruning: often a single LR model can replace a whole subtree
- Pruning proceeds bottom up: error for LR model at internal node is compared to error for subtree

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Lecture5 NeuralNet

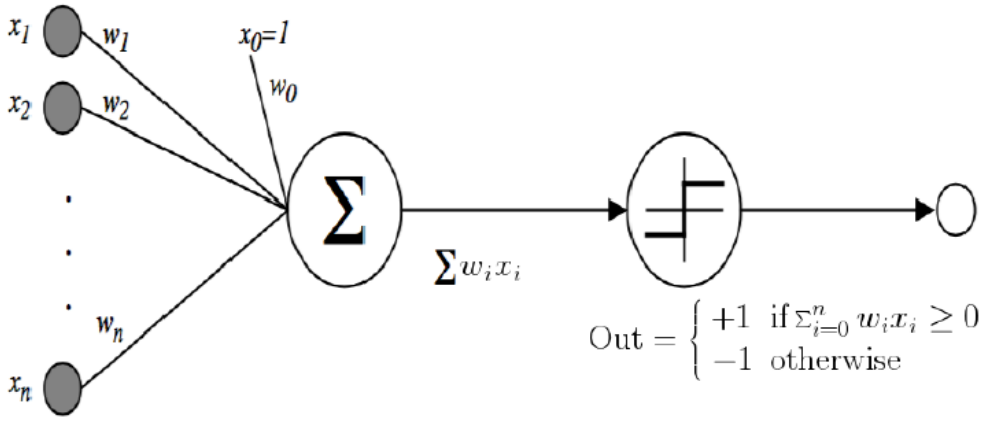
Aims

This lecture will enable you to describe and reproduce machine learning approaches to the problem of neural (network) learning. Following it you should be able to:

- describe Perceptrons and how to train them
- relate neural learning to optimization in machine learning
- outline the problem of neural learning
- derive the method of gradient descent for linear models
- describe the problem of non-linear models with neural networks
- outline the method of back-propagation training of a multi-layer perceptron neural network
- describe the application of neural learning for classification
- describe some issues arising when training deep neural networks

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Perceptron



Output o is thresholded sum of products of inputs and their weights:

$$o(x_1, \ldots, x_n) = \begin{cases} +1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Key idea:

Learning is "finding a good set of weights"

Perceptron learning is simply an iterative weight-update scheme:

$$w_i \leftarrow w_i + \Delta w_i$$

where the weight update Δw_i depends only on *misclassified* examples and is modulated by a "smoothing" parameter η typically referred to as the "learning rate".

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

- For example, let $\mathbf{x}_i$ be a misclassified positive example, then we have $y_i = +1$ and $\mathbf{w} \cdot \mathbf{x}_i < t$. We therefore want to find $\mathbf{w}'$ such that $\mathbf{w}' \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$, which moves the decision boundary towards and hopefully past x_i.
- This can be achieved by calculating the new weight vector as $\mathbf{w}' = \mathbf{w} + \eta \mathbf{x}_i$, where $0 < \eta \leq 1$ is the *learning rate* (again, assume set to 1). We then have $\mathbf{w}' \cdot \mathbf{x}_i = \mathbf{w} \cdot \mathbf{x}_i + \eta \mathbf{x}_i \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$ as required.

- The two cases can be combined in a single update rule:

$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

Algorithm Perceptron(D, η) // perceptron training for linear classification

Input: labelled training data D in homogeneous coordinates; learning rate η.
Output: weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
w ←0 // Other initialisations of the weight vector are possible
converged←false
while converged = false do
    converged←true
    for i = 1 to |D| do
        if y_i w · x_i ≤ 0 then          // i.e., ŷ_i ≠ y_i
            w←w + η y_i x_i
            converged←false  // We changed w so haven't converged yet
        end
    end
end
```

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Unfortunately, as a linear classifier perceptrons are limited in expressive power

So some functions not representable

- e.g., not linearly separable

For non-linearly separable data we'll need something else

However, with a fairly minor modification many perceptrons can be combined together to form one model

- *multilayer perceptrons*, the classic "neural network"

- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output can be discrete or real-valued
- Output can be a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Gradient Descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\mathbf{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient

$$\nabla E[\mathbf{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n} \right]$$

Gradient vector gives direction of *steepest increase* in error E

Negative of the gradient, i.e., *steepest decrease*, is what we want

Training rule:

$$\Delta \mathbf{w} = -\eta \nabla E[\mathbf{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \mathbf{w}\cdot\mathbf{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair $\langle \mathbf{x}, t\rangle$, where $\mathbf{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

Initialize each w_i to some small random value

Until the termination condition is met, Do

  Initialize each Δw_i to zero

  For each $\langle \mathbf{x}, t\rangle$ in $training\_examples$, Do

    Input the instance $\mathbf{x}$ to the unit and compute the output o

    For each linear unit weight w_i

      $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$

  For each linear unit weight w_i

    $w_i \leftarrow w_i + \Delta w_i$

该文档是极速PDF编辑器生成，
如果想去掉该提示，请访问并下载：
http://www.jisupdfeditor.com/

Batch mode Gradient Descent:
Do until satisfied

- Compute the gradient $\nabla E_D[\mathbf{w}]$
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D[\mathbf{w}]$

Incremental mode Gradient Descent:
Do until satisfied

- For each training example d in D
  - Compute the gradient $\nabla E_d[\mathbf{w}]$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_d[\mathbf{w}]$

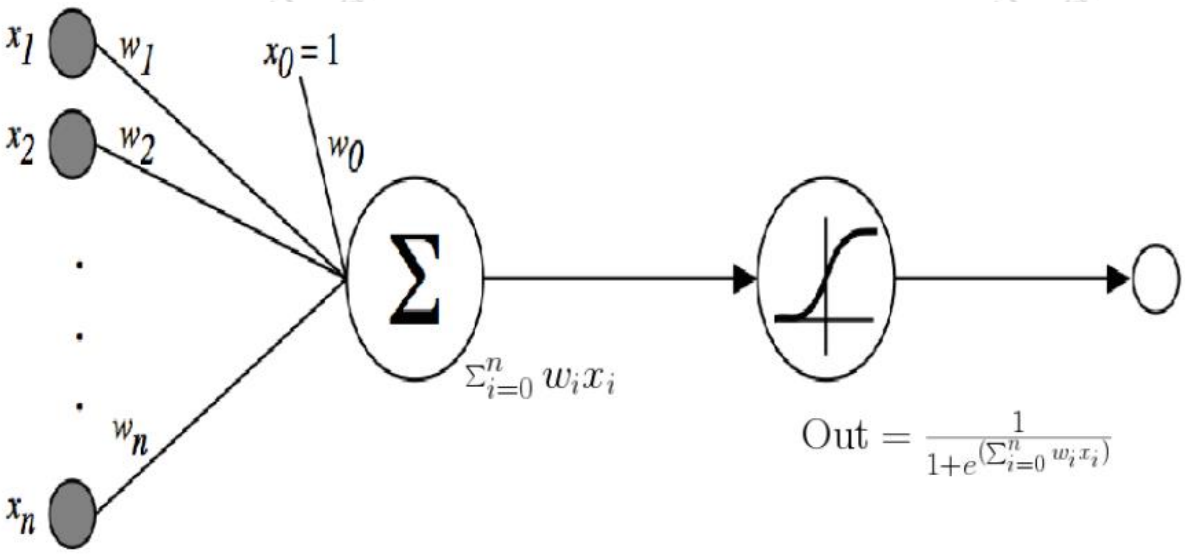
Batch:

$$E_D[\mathbf{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Incremental:

$$E_d[\mathbf{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental or *Stochastic Gradient Descent* (SGD) can approximate *Batch Gradient Descent* arbitrarily closely, if η made small enough



该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Why use the sigmoid function $\sigma(x)$?

$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

We can derive gradient descent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Start by assuming we want to minimize squared error $\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$ over a set of training examples D.

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\ \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d} 2(t_d - o_d)\ \frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d}(t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right) \\
&= -\sum_{d}(t_d - o_d)\ \frac{\partial o_d}{\partial net_d}\ \frac{\partial net_d}{\partial w_i}
\end{aligned}$$

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\mathbf{w} \cdot \mathbf{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

Initialize all weights to small random numbers.

Until satisfied, Do

For each training example, Do

Input the training example to the network and compute the network outputs

For each output unit k

$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$

For each hidden unit h

$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$

Update each network weight w_{ji}

$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

where

$\Delta w_{ji} = \eta \delta_j x_{ji}$

Will converge to a local, not necessarily global, error minimum

- May be many such local minima
- In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n - 1)$$

- Stochastic gradient descent using "mini-batches"

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Models can be very complex

- Will network generalize well to subsequent examples?
  - may *underfit* by stopping too soon
  - may *overfit* ...

Many ways to regularize network, making it less likely to overfit

- Add term to error that increases with magnitude of weight vector

$$E(\mathbf{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

- Other ways to penalize large weights, e.g., weight decay
- Using "tied" or shared set of weights, e.g., by setting all weights to their mean after computing the weight updates
- Many other ways ...

Deep Learning: Regularization

Deep networks can have millions or billions or parameters.

Hard to train, prone to overfit.

What techniques can help ?

Example: dropout

- for each unit u in the network, with probability p, "drop" it, i.e., ignore it and its adjacent edges during training
- this will simplify the network and prevent overfitting
- can take longer to converge
- but will be quicker to update on each epoch
- also forces exploration of different sub-networks formed by removing p of the units on any training run

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Neural networks for classification

Sigmoid unit computes output $o(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x})$

Output ranges from 0 to 1

Example: binary classification

$$o(\mathbf{x}) = \begin{cases} \text{predict class } 1 & \text{if } o(\mathbf{x}) \geq 0.5 \\ \text{predict class } 0 & \text{otherwise.} \end{cases}$$

Questions:

- what error (loss) function should be used ?
- how can we train such a classifier ?

该文档是极速PDF编辑器生成，
如果想去掉该提示,请访问并下载：
http://www.jisupdfeditor.com/

Minimizing square error (as before) does not work so well for classification

If we take the output $o(\mathbf{x})$ as the *probability* of the class of $\mathbf{x}$ being 1, the preferred loss function is the *cross-entropy*

$$-\sum_{d\in D} t_d \log o_d + (1-t_d)\log(1-o_d)$$

where:

$t_d \in \{0,1\}$ is the class label for training example d, and o_d is the output of the sigmoid unit, interpreted as the probability of the class of training example d being 1.

To train sigmoid units for classification using this setup, can use *gradient ascent* with a similar weight update rule as that used to train neural networks by gradient descent – this will yield the *maximum likelihood* solution.

Problem: in very large networks, sigmoid activation functions can *saturate*, i.e., can be driven close to 0 or 1 and then the gradient becomes almost 0 – effectively halts updates and hence learning for those units.

Solution: use activation functions that are non-saturating., e.g., "Rectified Linear Unit" or ReLu, defined as $f(x) = \max(0, x)$.

Problem: sigmoid activation functions are not zero-centred, which can cause gradients and hence weight updates become "non-smooth".

Solution: use zero-centred activation function, e.g., $tanh$, with range $[-1, +1]$. Note that $tanh$ is essentially a re-scaled sigmoid.

Derivative of a ReLu is simply

$$\frac{\partial f}{\partial x} = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise.} \end{cases}$$