Examples of Classifiers

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

May, 2015
Overview of prediction problems

Likelihood Ratio (Generative) Classifiers
- Linear Discriminant Analysis (LDA)
  [Quadratic Discriminant Analysis (QDA)]
- The Naive Bayes classifier

Discriminative classifiers
- Logistic Regression
  [The Perceptron algorithm]
- Classification and regression tree(s) (CART)
- The Nearest-Neighbor classifiers
  [Multi-layer Neural Networks]
Prediction problems – an overview

**Prediction=Supervised learning**

In supervised learning, the problem is **predicting** the value of an **output** (or **response** – typically in regression, or **label** – typically in classification) variable $Y$ from the values of some observed variables called **inputs** (or **predictors, features, attributes**) $(X_1, X_2, \ldots X_n) = X$. Typically we will consider that the input $X \in \mathbb{R}^n$.

Prediction problems are classified by the type of response $Y \in \mathcal{Y}$:

- **binary classification**: $Y \in \{-1, +1\}$
- **multiway classification**: $Y \in \{y_1, \ldots y_m\}$ a finite set $p$ objects
- **multilabel classification** $Y \subseteq \{y_1, \ldots y_m\}$ a finite set (i.e. each $X$ can have several labels)
- **regression**: $Y \in \mathbb{R}$
- **ranking**: $Y \in \mathcal{S}_p$ the set of permutations of
- **structured prediction** $Y \in \Omega_V$ the state space of a graphical model over a set of [discrete] variables $V$

In classification, the output $Y$ is called the **class variable** or **class label** will be sometimes denoted by $C$. These notes alternate between the $Y$ and $C$ notation.
Classification – the workflow

**Training phase**
- Get labeled data \( D = \{(x_1, c_1), (x_2, c_2), \ldots (x_n, c_n)\} \)
- Decide what type of classifier you want to build
- **Learn/estimate/(fit)** the classifier \( \phi \) from the data \( D \).
  - Here the goal is to find a \( \phi \) that classifies the data well
  - How to do it depends on what kind of classifier you have chosen. Most of the remaining slides are about this.

[**Validation phase** How good is really this \( \phi \)?]
- In this phase, you get a new data set \( D_v \), and check how well \( \phi \) classifies these new data. If \( \phi \) performs badly, return to training, with a new type of classifier.

**“Testing” phase** (i.e now you have a classifier, use it)
- Whenever a new, unlabeled example \( x \) comes in, output

\[
\hat{c}(x) = \phi(x)
\]

We call \( \hat{c}(x) \) the **predicted label** of \( x \).
Linear Discriminant Analysis (LDA)

A.k.a Likelihood ratio classification, where \( f_{X|C} \) are Normal, with the same variance \( \sigma^2 \) (or covariance matrix \( \Sigma \))

Fitting a linear classifier, first approach. (We are in the binary classification case)

- **Assume** each class is generated by a Normal distribution
  \[
P_X(x|+) = \mathcal{N}(x; \mu_+, \Sigma_+), \quad P_X(x|-) = \mathcal{N}(x; \mu_-, \Sigma_-) \quad \text{and} \quad P_Y(1) = p
  \]

- **Given** \( x \), what is the probability it came from class + ?
  \[
P_Y|x(+|x) = \frac{P_Y(1)P_X(x|+)}{P_Y(1)P_X(x|+) + P_Y(-)P_X(x|+)} \quad \text{and} \quad P_Y|x(-|x) = 1 - P_Y|x(+|x)
  \]

This formula is true whether the distributions \( P_X|C \) are normal or not.

- **We assign** \( x \) to the class with maximum posterior probability.
  \[
  \hat{c}(x) = \arg\max_{y \in \{\pm1\}} P_Y|x(y|x)
  \]

This too, holds true whether the distributions \( P_X|C \) are normal or not.

---

\( ^1 \)From here on, the output variable for classification will be denoted by \( C \).
Now we specialize to the case of normal class distribution. We assume in addition that \( \Sigma_+ = \Sigma_- = K\).

- **Decision rule:** \( \hat{c} = 1 \) iff \( P_{Y|X}(+|x) > P_{Y|X}(-|x) \)

- or equivalently iff

\[
0 \leq \phi(x) = \ln \frac{P_{Y|X}(+|x)}{P_{Y|X}(-|x)}
\]

\[
= \ln \frac{p}{1-p} - \frac{1}{2} \left[ x^T K x - 2 \mu_+^T K x + \mu_+^T K \mu_+ \right] - \frac{1}{2} \left[ x^T K x - 2 \mu_-^T K x + \mu_-^T K \mu_- \right] \tag{4}
\]

\[
= [K(\mu_+ - \mu_-)]^T x + \ln \frac{p}{1-p} + \mu_-^T K \mu_- - \mu_+^T K \mu_+ \tag{5}
\]

\[
= \beta^T x + \beta_0 \tag{6}
\]

- The above is a linear expression in \( x \), hence this classifier is called **(Fisher's) Linear Discriminant**

- Note that if we change the variables to \( x \leftarrow \sqrt{K} x, \mu_\pm \leftarrow \sqrt{K} \mu_\pm \), and if we shift the origin to \( \frac{\mu_+ + \mu_-}{2} \) (6) becomes

\[
2\mu_+^T x + \ln \frac{p}{1-p} \tag{7}
\]

This has a geometric interpretation
If we do not assume $\Sigma_+ = \Sigma_-$ then (3) is a quadratic function of $x$. Exercise Plot the curve $\phi(x) = 0$ in (3) for various data sets in two dimensions. What kind of curves do you observe? Can the decision region be bounded?

\[
\phi(x) = \ln \frac{p}{1-p} - \frac{1}{2} \left[ x - T \Sigma_+^{-1} x - 2 \mu_+^T \Sigma_+^{-1} x + \mu_+^T \Sigma_+^{-1} \mu_+ \right] \quad (8)
\]

\[
- \frac{1}{2} \left[ x^T \Sigma_-^{-1} x - 2 \mu_-^T \Sigma_-^{-1} x + \mu_-^T \Sigma_-^{-1} \mu_- \right] \quad (9)
\]
Let $\phi(x)$ be a classifier (not necessarily binary)

- $\phi(x)$ takes only a finite set of values
- The decision region associated with class $y$ is the region in $X$ space where $f$ takes value $y$, i.e. $D_y = \{x \in \mathbb{R}^n, \phi(x) = c\} = f^{-1}(y)$.
- The boundaries separating the decision regions are called decision boundaries.

- For a binary classifier, we have two decision regions, $D_+$ and $D_-$. By convention $\phi(x) = 0$ on the decision boundary.
- For binary classifier with real valued $\phi(x)$ (i.e. $\hat{c} = \text{sgn} \phi(x)$) we define $D_+ = \{x \in \mathbb{R}^n, \phi(x) > 0\}$, $D_- = \{x \in \mathbb{R}^n, \phi(x) < 0\}$ and the decision boundary $\{x \in \mathbb{R}^n, \phi(x) = 0\}$
The Naive Bayes classifier

- **Idea:** Assume the distribution in each class is a product

\[ P_{X|C=c} = \prod_{j=1}^{m} P_{X_j|C=c} \]  

(10)

In other words, the \( m \) attributes are independent given the class \( y \).

- Then, \( P_{C|X} \) is given by Bayes’ rule as follows

\[
P_{C|X}(+1|x) = \frac{P_C(1)P_{X|C}(x|+1)}{P_C(1)P_{X|C}(x|+1) + P_C(-1)P_{X|C}(x|-1)}
\]

(11)

- The advantage of this classifier is in its simplicity
  1. for each class \( c \), and each attribute \( j \) estimate \( P_{X_j|C=c} \)
  2. for each class \( c \), estimate \( P_C(c) \).

Done (works for more than 2 classes as well!)

- For binary classification (as in LDA), we obtain

\[
\phi(x) = \ln \frac{P_{C|X}(+1|x)}{P_{C|X}(-1|x)} = \ln \frac{p}{1-p} + \sum_{j=1}^{n} \ln \frac{P_{X_j|C}(x_j|+1)}{P_{X_j|C}(x_j|-1)}
\]

(12)
Example – Naive Bayes for Discrete variables

**Input/output**  \( X \in \{0, 1\}^{8 \times 8} \) digit images, \( C \in \{0, 1, 3, \ldots 9\} \)

**Data**  \( n \) labeled images of digits \( D \)

**Model**  \( P_{X|C} = P_{X_1|C}P_{X_2|C} \ldots P_{X_{64}|C} \) (where \( X_1 \) is the top-left pixel and \( X_{64} \) is the bottom-right pixel)

with \( P_{X_j|C=c} = \text{Bernoulli}(\theta_{jc}) \) for \( j = 1 : 64, \ c = 0 : 9 \).

[Likelihood] of the point \( x = (1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, \ldots 1, 0), \ c = 3 \) is

\[
P_{X|C}(x, c) = \theta_{1,3}\theta_{2,3} \ldots \theta_{7,3}(1 - \theta_{8,3}) \ldots \theta_{63,3}(1 - \theta_{64,3})
\]

“Training”  Estimate \( \theta_{j,c} \) by Maximum Likelihood for \( j = 1 : 64, \ c = 0 : 9 \), e.g.

\[
\theta_{1,3} = \frac{\text{#pixel } 1 = 1 \text{ when } c = 3}{\# c=3} \quad \text{Estimate class probabilities } P_C \text{ by Maximum Likelihood (for example } P_C(3) = n_3/n \)

“Test”  Get new digit image \( x^{\text{new}} = (0, 0, 1, 1, 1, 1, 0, 0, \ldots 1, 0) \) with no label.

- Compute posterior probabilities of class \( c = 0 : 9 \)

\[
\ln P_{C=c|X}(x^{\text{new}}) = \ln(1 - \theta_{1,c}) + \ln(1 - \theta_{2,c}) + \ln \theta_{3,c} + \ldots + \ln(1 - \theta_{64,c}) + \ln P_C(c)
\]

\[
= \sum_{j=1}^{64} \left[ x_j^{\text{new}} \ln \theta_{j,c} + (1 - x_j^{\text{new}}) \ln(1 - \theta_{j,c}) \right] + \ln P_C(c)
\]

Output  the label \( c(x^{\text{new}}) = \arg\max_{c=0:9} \ln P_{C=c|X}(x^{\text{new}}) \) of the most probable class and confidence \( P_{C=c}(x^{\text{new}}|X(x^{\text{new}})) \)
Discriminative vs generative classifiers

- Likelihood Ratio classifiers are also called **generative classifiers**
  - because they estimate a separate model $f_{X|C=c}$ for each class from the examples with label $c$
  - this is done independently (and separately) of the examples from other classes

- **Discriminative** classifiers estimate decision boundaries directly
  - there are MANY types of discriminative classifiers, and we will study a few of the best known ones
Logistic Regression

Fits a linear probabilistic classifier, directly. Let $\phi(x) = \beta^T x$ model the log odds of class 1

$$\phi(x) = \frac{P(Y = 1|X)}{P(Y = -1|X)}$$

Then

- $\hat{c} = 1$ iff $P(Y = 1|X) > P(Y = -1|X)$
- just like in the previous case! so what’s the difference?
Logistic Regression

Fits a linear probabilistic classifier, directly. Let $\phi(x) = \beta^T x$ model the log odds of class 1

$$\phi(x) = \frac{P(Y = 1|X)}{P(Y = -1|X)}$$

(13)

Then

- $\hat{c} = 1$ iff $P(Y = 1|X) > P(Y = -1|X)$
  - just like in the previous case! so what’s the difference?
  - Answer: We don’t assume each class is Gaussian, so we are in a more general situation than LDA

- What is $p(x) = P(Y = 1|X = x)$ under our linear model?

$$\ln \frac{p}{1 - p} = f, \quad \frac{p}{1 - p} = e^f, \quad p = \frac{e^f}{1 + e^f}, \quad 1 - p = \frac{1}{1 + e^f}$$

(14)

An alternative “symmetric” expression for $p, 1 - p$ is

$$p = \frac{e^{f/2}}{e^{f/2} + e^{-f/2}}, \quad 1 - p = \frac{e^{-f/2}}{e^{f/2} + e^{-f/2}}.$$  

(15)
Estimating the parameters by Max Likelihood

- Denote $c^* = (1 - c)/2 \in \{0, 1\}$
- The likelihood of a data point is $P_{C|X}(c|x) = \frac{e^{c^* \phi(x)}}{1 + e^{\phi(x)}}$
- The log-likelihood is $l(\beta; x) = c^* \phi(x) - \ln(1 + e^{\phi(x)})$
- $\frac{\partial l}{\partial f} = c^* - \frac{1}{1 + e^{\phi(x)}}$
  This is a scalar, and $\text{sgn} \frac{\partial l}{\partial f} = c$
- We have also $\frac{\partial \phi(x)}{\partial \beta} = x$
- Now, the gradient of $l$ w.r.t the parameter vector $\beta$ is
  \[ \frac{\partial l}{\partial \beta} = \frac{\partial l}{\partial f} \frac{\partial f}{\partial \beta} = \left( c^* - \frac{1}{1 + e^{\phi(x)}} \right) x \] (16)
  Interpretation: The infinitesimal change of $\beta$ to increase log-likelihood for a single data point is along the direction of $x$, with the sign of $y$
Estimating the parameters by Max Likelihood

- Denote $c^* = (1 - c)/2 \in \{0, 1\}$
- The likelihood of a data point is $P_{C|X}(c|x) = \frac{e^{c^*\phi(x)}}{1 + e^{\phi(x)}}$
- The log-likelihood is $l(\beta; x) = c^*\phi(x) - \ln(1 + e^{\phi(x)})$
- $\frac{\partial l}{\partial f} = c^* - \frac{1}{1 + e^{\phi(x)}}$
  This is a scalar, and $\text{sgn} \frac{\partial l}{\partial f} = c$
- We have also $\frac{\partial \phi(x)}{\partial \beta} = x$
- Now, the gradient of $l$ w.r.t the parameter vector $\beta$ is
  \[
  \frac{\partial l}{\partial \beta} = \frac{\partial l}{\partial f} \frac{\partial f}{\partial \beta} = \left( c^* - \frac{1}{1 + e^{\phi(x)}} \right) x
  \]
  Interpretation: The infinitesimal change of $\beta$ to increase log-likelihood for a single data point is along the direction of $x$, with the sign of $y$
- Log-likelihood of the data set $D$
  \[
  l(\beta; D) = \frac{1}{N} \sum_{i=1}^{N} l(\beta; (x^i, y^i))
  \]
- The optimal $\beta$ maximizes $l(\beta; D)$ and therefore
  \[
  \frac{\partial l(\beta; D)}{\partial \beta} = \frac{1}{N} \sum_{i=1}^{N} \left( y^i - \frac{1}{1 + e^{f(x^i)}} \right) x^i = 0
  \]
- Unfortunately, (18) does not have a closed form solution! We maximize the (log)likelihood by iterative methods (e.g. gradient ascent) to obtain the $\beta$ of the classifier.
  - this iterative estimation converges asymptotically
The Perceptron algorithm

Yet another linear classifier.
Define $\phi(x) = \beta^T x$ and find $\beta$ that classifies all the data correctly (when possible).

Perceptron Algorithm

Input labeled training set $\mathcal{D}$

Initialize $\beta = 0$, for all $i$, $x^i \rightarrow \frac{x^i}{||x^i||}$ (normalize the inputs)

Repeat until no more mistakes

for $i = 1 : N$

1. if $\text{sgn}(\beta^T x^i) \neq y^i$ (a mistake)
   $\beta \leftarrow \beta + y^i x^i$

(Other variants exist)
The perceptron algorithm and linearly separable data

- $\mathcal{D}$ is linearly separable iff there is a $\beta_*$ so that $\text{sgn} \beta_*^T x^i = y^i$ for all $i = 1, \ldots, N$. If one such $\beta_*$ exists, then there are an infinity of them.

**Theorem**

Let $\mathcal{D}$ be a linearly separable data set, and define

$$
\gamma = \min_i \frac{|\beta_*^T x^i|}{||\beta_*|| ||x^i||}.
$$

(19)

Then, the number of mistakes made by the **Perceptron algorithm** is at most $1/\gamma^2$.

- Note that if we scale the examples to have norm 1, then $\gamma$ is the minimum distance to the hyperplane $\beta_*^T x = 0$ in the data set.

**Exercise** Show that if $\mathcal{D}$ is linearly separable, the scaling $x^i \rightarrow \frac{x^i}{||x^i||}$ leaves it linearly separable.

- If $\mathcal{D}$ is not linearly separable, the algorithm oscillates indefinitely.
Classification (And Regression) Trees (CART)

- A **classification tree** or (**decision tree**) is built recursively by splitting the data with hyperplanes parallel to the coordinate axes.
  - At each split, try to separate + examples from − examples as well as possible.
  - Eventually, all the regions will be “pure”, i.e. will contain examples from one class only.
- Classification trees can be used in multiway classification as well (there we try to create a pure region on at least one side of the split)
- A **regression tree** uses the same principle for regression here we try to obtain regions where the outputs are nearly the same
Hierarchical partitions

- A hierarchical partition $\mathcal{T}$ of $\mathbb{R}^n$ is a set of regions $\{R_q\}$, so that $\mathbb{R}^n = \bigcup_q R_q$ and between any two $R_q, R_{q'}$ we have either

$$R_q \cap R_{q'} = \emptyset, \text{ or } R_q \subset R_{q'} \text{ or } R_{q'} \subset R_q. \quad (20)$$

- In a CART, the partitions are usually chosen to be axis-aligned, i.e.

$$R_q = \{x \mid \pm x_{j_1} > \tau_1, \pm x_{j_2} > \tau_2, \ldots \pm x_{j_l} > \tau_l\} \text{ where } > \text{ stands for one of } > \text{ or } \geq,$$

so that the union of all regions covers $\mathbb{R}^n$.

- The number of inequalities $l$ defining the region is called the level of the region.

- $R_q$ is a leaf of $\mathcal{T}$ iff there is no other $R_{q'}$ included in it.

Example (A hierarchical partition with 3 levels over $\mathbb{R}^2$)

| Level 1: | $R_1 = \{x \mid x_2 > 3\}$, |
| | $R_2 = \{x \mid x_2 \leq 3\}$ |
| Level 2: | $R_3 = \{x \mid x_2 > 3, x_1 \geq -2\}$, |
| | $R_4 = \{x \mid x_2 > 3, x_1 < -2\}$, |
| | $R_5 = \{x \mid x_2 \leq 3, x_1 > 0\}$, |
| | $R_6 = \{x \mid x_2 \leq 3, x_1 \leq 0\}$ |
| Level 3: | $R_7 = \{x \mid x_2 > 3, x_1 \geq -2, x_1 < 4\}$, |
| | $R_8 = \{x \mid x_2 > 3, x_1 \geq 4\}$, |
| | $R_9 = \{x \mid x_2 < 3, x_1 \geq 1\}$ |
| | $R_{10} = \{x \mid x_2 \leq 3, x_1 \leq 0, x_2 > -1\}$, |
| | $R_{11} = \{x \mid x_2 \leq -1, x_1 \leq 0\}$, |
| | $R_{12} = \{x \mid x_2 < 3, x_1 > 0, x_1 < 1\}$ |

The leaves are $R_4, R_7, \ldots R_{12}$. Not all leaves are at the same level; for example $R_4$ is at level 2.
CARTs are piecewise constant functions over leaf sets of axis-aligned hierarchical partitions, i.e.

\[ \phi(x) = y_q \text{ whenever } x \in R_q, \text{ with } R_q \text{ a leaf of a hierarchical partition.} \]

Thus, \( f \) is defined by the values \((y_q)\) and the set of regions \((R_q)\). This inclusion hierarchy of a hierarchical partition can be represented by a tree, in which \( R_q, R_{q'} \) are connected by an edge iff one region contains the other. The root of the tree is the whole of \( \mathbb{R}^n \). The leaves are the leaves of this tree. The tree corresponding to Example 2 is

\[ \begin{array}{c}
\mathbb{R}^n \\
R_1 \\
R_3 \\
R_7 \\
R_2 \\
R_4 \\
R_8 \\
R_5 \\
R_9 \\
R_12 \\
R_6 \\
R_{10} \\
R_{11} \\
\end{array} \]
Natural ways to set $y_q$ based on the data, once the partition $\mathcal{T}$ has been fixed:

- denote $Y_q = \{y^i \mid x^i \in R_q, \ i = 1 : N\}$ the set of labels at a leaf $R_q$
- **Regression** $y_q = \text{average of } Y_q$
- **Classification** $y_q = \text{majority label of } Y_q$

- a leaf $R_q$ is **pure** if all labels are the same, i.e. if $|C_q| = 1$

- criteria for the **(im)purity** of a leaf $R_q$
  - **Regression** impurity = sample variance of $Y_q$
  - **Classification** let $p_q$ be the frequency of $y_q$ in $Y_q$

$$\text{impurity} = \begin{cases} 
\text{Misclassification error} & 1 - p_q \\
\text{Gini} & p_q(1 - p_q) \\
\text{Entropy} & p_q \ln p_q + (1 - p_q) \ln(1 - p_q)
\end{cases} \quad (21)$$

These measures generalize naturally to the multiclass setting.
“Learning” a CART

A standard algorithm for building a decision tree works recursively in top-down fashion.

**Input** Training set $\mathcal{D}$ of size $N$

**Initialize** with a tree with only one region, that contains all the data

Repeat until all leaves are pure (or until desired purity is attained in all leaves)

1. Find the “optimal” split over all leaves $R_q$ and all possible splits of $R_q$.
   “Optimal” is defined in terms on purity (e.g split the least pure leaf, find the split that makes one of the new leaves pure)

2. Perform the “optimal” split and add the two new leaves to the tree

This is a greedy algorithm. Sometimes, trees obtained this way are pruned back to smaller sizes.
A decision tree over $D$ is not unique

Same dataset $D$, two different trees. Both classify the sample $D$ perfectly.

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
<th>differences of $T_1$, $T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

But they produce different decision regions.
The Nearest-Neighbor predictor

- **Main Idea** The label of a point $x$ is assigned as follows:
  1. find the example $x^i$ that is nearest to $x$ (in Euclidean distance)
  2. assign $x$ the label $y^i$

- Practically, one uses the $K$ nearest neighbors of $x$ (with $K = 3, 5$ or larger), then
  - for classification $\phi(x) = \text{the most frequent label among the } K \text{ neighbors}$
    (well suited for multiclass)
  - for regression $\phi(x) = \frac{1}{K} \sum_{i \text{ neighbor of } x} y^i = \text{mean of neighbors’ labels}$

- **No parameters to estimate!** (But all data must be stored)
Two-layer Neural Networks

- The activation function (a term borrowed from neuroscience) is any continuous, bounded and strictly increasing function on $\mathbb{R}$. Almost universally, the activation function is the logistic (or sigmoid)

$$g(z) = \frac{1}{1 + e^{-z}}$$  \hspace{1cm} (22)

because of its nice additional computational and statistical properties.

- We build a two-layer neural network in the following way:

  Inputs $x_k$ $k = 1 : n$
  Bottom layer $z_j = g(w_j^T x)$ $j = 1 : m$, $w_j \in \mathbb{R}^n$
  Top layer $f = g(\beta^T z)$ $\beta \in \mathbb{R}^m$
  Output $f \in [0, 1]$

In other words, the neural network implements the function

$$\phi(x) = g \left( \sum_{j=1}^{m} \beta_j z_j \right) = g \left( \sum_{j=1}^{m} \beta_j g(\sum_{j} w_{kj}x_k) \right)$$  \hspace{1cm} (23)

- in classification $\phi(x) \in [0, 1]$ is interpreted as the probability of the + class.
- in regression the sigmoid of the top layer is omitted. We say that the top layer is a linear layer.

$$\phi(x) = \sum_{j=1}^{m} \beta_j z_j = \sum_{j=1}^{m} \beta_j g(\sum_{j} w_{kj}x_k) \in (-\infty, \infty)$$  \hspace{1cm} (24)

Note that this is just a linear combination of logistic functions.

2In neural net terminology, each variable $z_j$ is a unit, the bottom layer is hidden, while top one is visible, and the units in this layer are called hidden/visible units as well. Sometimes the inputs are called input units; imagine neurons or individual circuits in place of each $x, y, z$ variable.
Multi-layer neural networks

The construction can be generalized recursively to arbitrary numbers of layers. Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function $g$. Let $x \equiv x^{(0)}$, $y \equiv x^{(L)}$, $n_0 = n$, $n_L = 1$ and define the recursion:

$$x_j^{(l)} = g \left( (w_j^{(l)})^T x^{(l-1)} \right), \text{ for } j = 1 : n_l$$

(25)

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the output of layer $l$ of the network. As before, the sigmoid of the last layer may be omitted.
Are multiple layers necessary?

- 1990’s: NO
- 2000’s: YES

- A theoretical result

**Theorem (Cybenko, \(\approx 1986\))**

Any continuous function from \([0, 1]^n\) to \(\mathbb{R}\) can be approximated arbitrarily closely by a linear output, two layer neural network defined in (24) with a sufficiently large number of hidden units \(m\).

- A practical result

**Deep Learning**

*Deep learning* = multi-layer neural net

- So, what is new?
  - small variations in the “units”, e.g. switch stochastically w.p. \(g(w^T x^i)\) *(Restricted Boltzmann Machine)*
  - training method auto-encoders vs. back-propagation (we will return to this when we talk about training predictors)