# Non-linear Classification

Anita Faul

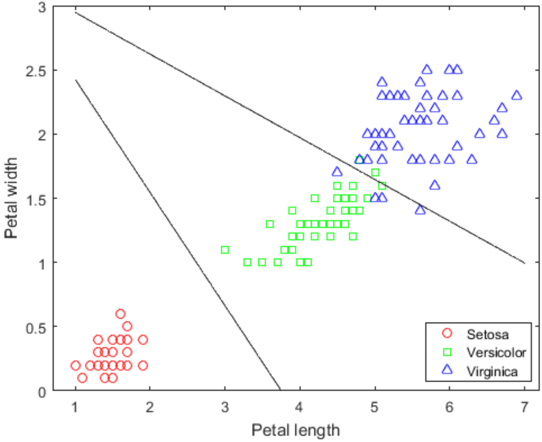
Laboratory for Scientific Computing, University of Cambridge

Non-linear classification

- Quadratic Discriminant Analysis (QDA)
- Kernel trick
- $k$ Nearest Neighbours ($k$-NN)
- Decision trees
- Neural Networks
- Boosting
- Cascades

# Classification

So far we assumed that the data can be separated by a straight line. However, this assumption of linear classification is very restrictive.



Probability vs. Likelihood

- subtle difference
- Probability answers the question: How probable is it that a sample of class $C_i$ has these features?
- Likelihood answers the question: How likely is it that a sample with these features belongs to class $C_i$?

# Probability vs. Likelihood

Probability vs. Likelihood

- Let the features of samples in class $C_i$ be normally distributed with mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\Sigma}_i$. Let $\mathbf{v}$ be a feature vector.
- Probability of $\mathbf{v}$ given that it is in class $C_i$:

$$p(\mathbf{v}|\mathbf{v} \in C_i) = \frac{1}{\sqrt{(2\pi)^M |\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{v}-\boldsymbol{\mu}_i)\right)$$

- Likelihood of $\mathbf{v} \in C_i$ given that we know its features $\mathbf{v}$:

$$\mathcal{L}(\mathbf{v} \in C_i|\mathbf{v}) = \frac{1}{\sqrt{(2\pi)^M |\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{v}-\boldsymbol{\mu}_i)\right)$$

- Same formula, different interpretation.

# Likelihood ratio

- Assign $\mathbf{v}$ to class $C_0$ if the likelihood of it belonging to $C_0$ is larger than the likelihood of it belonging to $C_1$.
- The boundary between the classes is given by

$$\mathcal{L}(\mathbf{v} \in C_0|\mathbf{v}) = \mathcal{L}(\mathbf{v} \in C_1|\mathbf{v}).$$

- Often expressed as likelihood ratio

$$\frac{\mathcal{L}(\mathbf{v} \in C_0|\mathbf{v})}{\mathcal{L}(\mathbf{v} \in C_1|\mathbf{v})} = 1.$$

# Log of likelihood ratio

$$\frac{\mathcal{L}(\mathbf{v} \in C_0|\mathbf{v})}{\mathcal{L}(\mathbf{v} \in C_1|\mathbf{v})} = \frac{\sqrt{(2\pi)^M|\boldsymbol{\Sigma}_1|}\exp\left(-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_0^{-1}(\mathbf{v}-\boldsymbol{\mu}_0)\right)}{\sqrt{(2\pi)^M|\boldsymbol{\Sigma}_0|}\exp\left(-\frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\mathbf{v}-\boldsymbol{\mu}_1)\right)} = 1.$$

Take logarithm

$$\begin{aligned}\log\frac{\mathcal{L}(\mathbf{v} \in C_0|\mathbf{v})}{\mathcal{L}(\mathbf{v} \in C_1|\mathbf{v})} &= \frac{1}{2}\log\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|} - \frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_0^{-1}(\mathbf{v}-\boldsymbol{\mu}_0) \\ &\quad + \frac{1}{2}(\mathbf{v}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\mathbf{v}-\boldsymbol{\mu}_1) \\ &= \frac{1}{2}\left(\log\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|} + \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0\right) \\ &\quad + \mathbf{v}^T\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1\right) + \frac{1}{2}\mathbf{v}^T\left(\boldsymbol{\Sigma}_1^{-1} - \boldsymbol{\Sigma}_0^{-1}\right)\mathbf{v} = 0\end{aligned}$$

# Quadratic Discriminant Analysis (QDA)

Thus the boundary between classes is given by a quadratic of the form

$$\mathbf{v}^T \mathbf{A} \mathbf{v} + \mathbf{v}^T \mathbf{b} + c = 0$$

with

$$\begin{aligned}
\mathbf{A} &= \frac{1}{2}\left(\mathbf{\Sigma}_1^{-1} - \mathbf{\Sigma}_0^{-1}\right), \\
\mathbf{b} &= \mathbf{\Sigma}_0^{-1}\boldsymbol{\mu}_0 - \mathbf{\Sigma}_1^{-1}\boldsymbol{\mu}_1, \\
c &= \frac{1}{2}\left(\log\frac{|\mathbf{\Sigma}_1|}{|\mathbf{\Sigma}_0|} + \boldsymbol{\mu}_1^T\mathbf{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\mathbf{\Sigma}_0^{-1}\boldsymbol{\mu}_0\right)
\end{aligned}$$

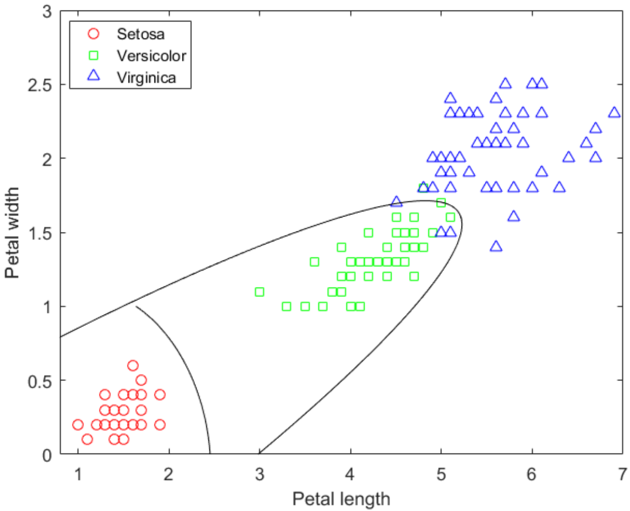
This is the *Quadratic Discriminant Analysis*.

# QDA

- In the special case of two features the boundaries are conic sections, that is a line, circle, ellipse, parabola or hyperbola.
- Implemented in Matlab as

```
ClassificationDiscriminant.fit
```

# QDA



Setosa
Versicolor
Virginica
Petal width
Petal length
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7

# LDA revisited

If $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_1 = \mathbf{\Sigma}$, then

$$
\begin{aligned}
\mathbf{A} &= 0, \\
\mathbf{b} &= \mathbf{\Sigma}^{-1}\left(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1\right), \\
c &= \frac{1}{2}\left(\boldsymbol{\mu}_1^T \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_0\right).
\end{aligned}
$$

This is exactly the equation of the line in the *Linear Discriminant Analysis*.

- In the quadratic

$$\mathbf{v}^T\mathbf{A}\mathbf{v} + \mathbf{v}^T\mathbf{b} + c = 0$$

  the matrix $\mathbf{A}$ is symmetric.
- Assume $M = 2$ and

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

- The boundary can be written as

$$A_{11}v_1^2 + 2A_{12}v_1v_2 + A_{22}v_2^2 + b_1v_1 + b_2v_2 + c = 0.$$

# Kernel trick

- This can be written as

$$\begin{pmatrix} A_{11} & 2A_{12} & A_{22} & b_1 & b_2 \end{pmatrix} \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2^2 \\ v_1 \\ v_2 \end{pmatrix} + c = 0.$$

- Letting $\mathbf{w}^T = \begin{pmatrix} A_{11} & 2A_{12} & A_{22} & b_1 & b_2 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2^2 \\ v_1 \\ v_2 \end{pmatrix}$, this describes a linear boundary in a five-dimensional feature space.

$$\mathbf{w}^T \mathbf{y} + c = 0.$$

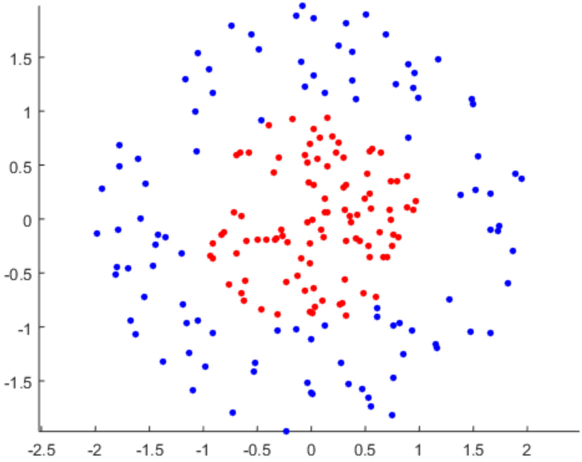
# Kernel trick

- We have augmented the feature space by three artificial features

$$v_1^2, \qquad v_1 v_2, \qquad v_2^2.$$

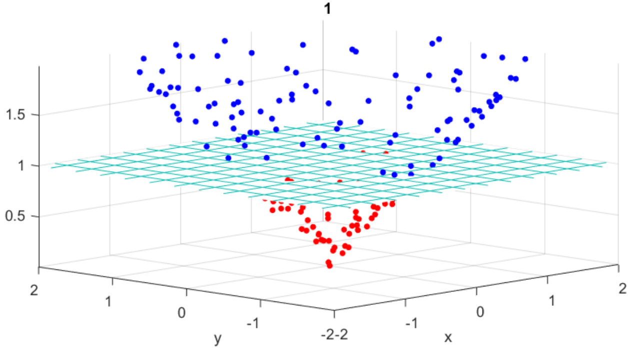
- This is known as the *Kernel trick*.
- The features are transformed to a high-dimensional feature space where the classes are linearly separable.
- (In practice this transformation is never performed.)

# Kernel trick example




Set $v_3 = \sqrt{v_1^2 + v_2^2}$

# Kernel trick example




- The data can be separated by the plane $v_3 = 1$.
- $v_3^2 = v_1^2 + v_2^2 = 1$ is the unit circle in the $v_1, v_2$-plane.

Recall that the SVM classifies according to the sign of

$$\hat{\mathbf{w}}^T\hat{\mathbf{v}} = \sum_{n=1}^{N} \alpha_n c_n \hat{\mathbf{v}}_n^T \hat{\mathbf{v}} = \sum_{n=1}^{N} \alpha_n c_n \left(1 + \mathbf{v}_n^T \mathbf{v}\right) = -b + \sum_{n=1}^{N} \alpha_n c_n \mathbf{v}_n^T \mathbf{v},$$

where the bias $b$ is given by $-\sum_{n=1}^{N} \alpha_n c_n$ and where $c_i = \pm 1$ is the class label of sample $\mathbf{v}_i$. The $\alpha_i \geq 0$ for $i = 1, \ldots, N$ maximize the function

$$L(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{i=1}^{N} \sum_{n=1}^{N} \alpha_i \alpha_n c_i c_n \left(1 + \mathbf{v}_i^T \mathbf{v}_n\right).$$

# Kernel SVM

- Let $\phi : \mathbb{R}^M \to \mathbb{R}^{\hat{M}}$ be the mapping from the original $M$ dimensional feature space to the higher $\hat{M}$ dimensional feature space.
- We need to maximize

$$
\begin{aligned}
L(\boldsymbol{\alpha}) &= \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{i=1}^{N} \sum_{n=1}^{N} \alpha_i \alpha_n c_i c_n \left(1 + \phi(\mathbf{v}_i)^T \phi(\mathbf{v}_n)\right) \\
&= \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{i=1}^{N} \sum_{n=1}^{N} \alpha_i \alpha_n c_i c_n \left(1 + k(\mathbf{v}_i, \mathbf{v}_n)\right).
\end{aligned}
$$

- $k : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ defined as $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$ is the *Kernel function*.

# Kernel SVM

- The classification is according to the sign of

$$-b + \sum_{n=1}^{N} \alpha_n c_n \phi(\mathbf{v}_n)^T \phi(\mathbf{v}) = -b + \sum_{n=1}^{N} \alpha_n c_n k(\mathbf{v}_n, \mathbf{v}).$$

- The mapping $\phi$ never has to be evaluated only the kernel function $k$.

# Kernel SVM

To summarize

- Training: Maximize

$$L(\boldsymbol{\alpha}) \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{i=1}^{N} \sum_{n=1}^{N} \alpha_i \alpha_n c_i c_n \left(1 + k(\mathbf{v}_i, \mathbf{v}_n)\right).$$

- Classifying: Evaluate

$$-b + \sum_{n=1}^{N} \alpha_n c_n k(\mathbf{v}_n, \mathbf{v}).$$

# Kernel trick

- The kernel trick can be applied to all linear classification methods to separate data which is not linearly separable.
- All methods reduce to evaluations of the kernel function $k$.

# Kernels

- The kernel function was derived from a higher dimensional inner product.
- It is a measure of the relation ship between different samples.
- A new sample is classified according to how similar with regards to this measure it is to the training samples, that is the support vectors.
- Other kernels are defined.

# Kernels

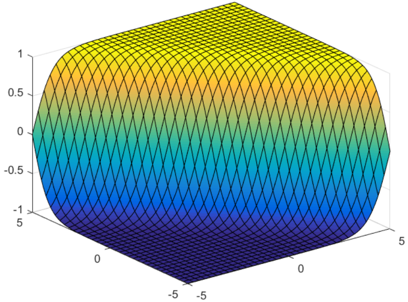
Linear (trivial) kernel $k(\mathbf{x}, \mathbf{y}) = a\mathbf{x}^T\mathbf{y} + c$.

Quadratic kernel $k(\mathbf{x}, \mathbf{y}) = (a\mathbf{x}^T\mathbf{y} + c)^2$.

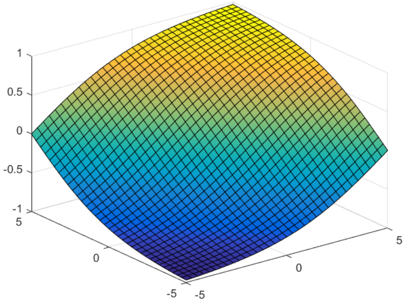
Polynomial kernel (of degree $d$) $k(\mathbf{x}, \mathbf{y}) = (a\mathbf{x}^T\mathbf{y} + c)^d$.

Hyperbolic tangent (Sigmoid) kernel $k(\mathbf{x}, \mathbf{y}) = \tanh(a\mathbf{x}^T\mathbf{y} + c)$.

# Kernels




(a) $a = 1, \mathbf{y} = (1\ 1), c = 0$




(b) $a = 0.2, \mathbf{y} = (1\ 1), c = 0$

Figure: Hyperbolic tangent kernel.

# Kernels

Gaussian kernel $k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$.

Exponential kernel $k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|}{2\sigma^2}\right)$.

Laplacian kernel $k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|}{\sigma}\right)$.

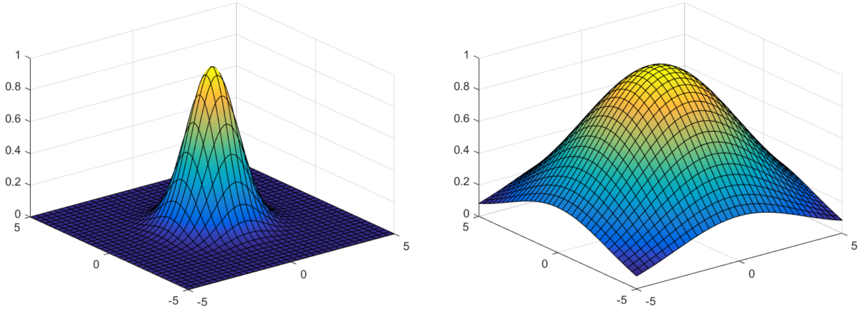
Multiquadric kernel $\sqrt{\|\mathbf{x}-\mathbf{y}\|^2 + c^2}$.

Inverse multiquadric kernel $\frac{1}{\sqrt{\|\mathbf{x}-\mathbf{y}\|^2 + c^2}}$.

Rational quadratic kernel $1 + \frac{\|\mathbf{x}-\mathbf{y}\|^2}{\|\mathbf{x}-\mathbf{y}\|^2 + c}$.

Thin plate spline kernel $\|\mathbf{x}-\mathbf{y}\|^2 \log\|\mathbf{x}-\mathbf{y}\|$.

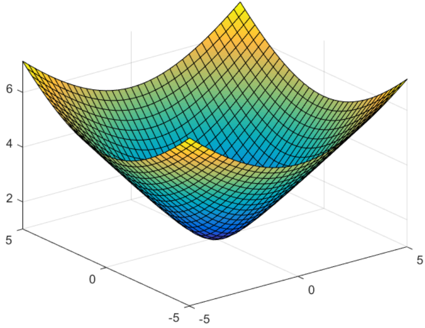
# Kernels



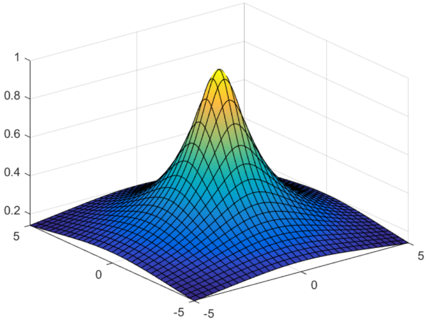

(a) $\mathbf{y} = (0\,0), \sigma^2 = 1$ (b) $\mathbf{y} = (0\,0), \sigma^2 = 10$

Figure: Gaussian kernel.

# Kernels



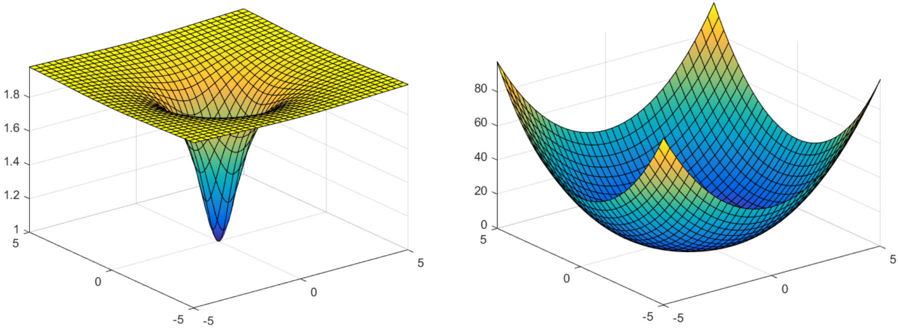

(a) Multiquadric kernel, $c = 1$.




(b) Inverse multiquadric kernel, $c = 1$.

Figure: Radial basis function kernels

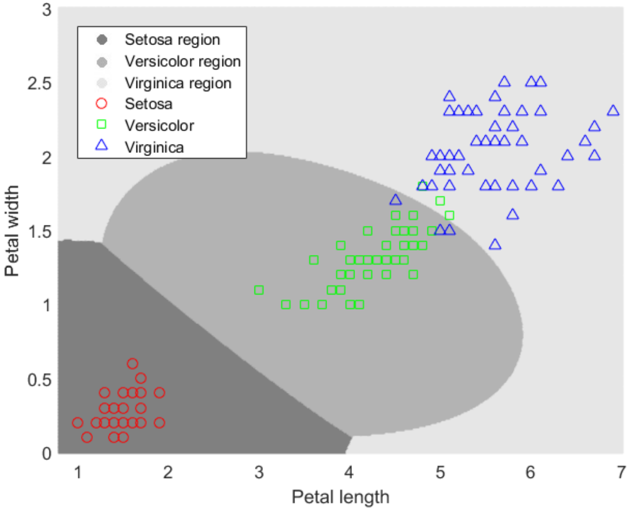
# Kernels




(a) Rational quadratic kernel, $c = 1$. (b) Thin plate spline kernel.

Figure: Radial basis function kernels

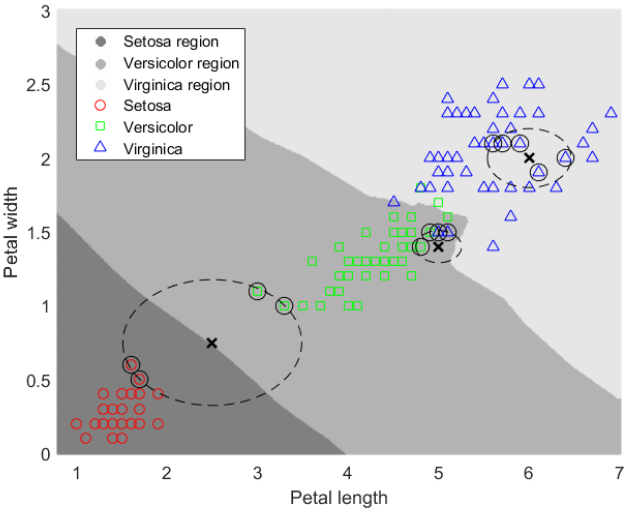
# Example RBF SVM



# $k$ Nearest Neighbours

- Classification is done by checking how similar a new sample is to the training samples.
- A simple idea is to classify a new sample according to the majority of classes its neighbours belong.
- The technique is *$k$ Nearest Neighbours ($k$-NN)*.

# $k$ Nearest Neighbours



# Decision Trees

*Decision Trees* are binary trees consisting of

- *non-terminal nodes* having two branches,
- *terminal nodes* or *leaves* which are assigned a class.

A sample enters the tree at the *root* node at the top. At each node, a decision is made whether the value of a particular feature is larger or smaller than a threshold. The sample traverses the tree down to a leaf and assigned that class.

- Decision trees are grown recursively from a training set.
- At each node, the set of training samples which reached that node is split in two.
- Let $t = P, L$, or $R$ refer to quantities relating to the parent, left or right child node respectively.
- $N_P$ is the number of samples reaching the parent node, while $N_L$ and $N_R$ are the number of samples reaching the left and right child nodes.

# Decision Trees

Let $N_{tk}$, be the number of samples in class $k$ reaching node $t$. $p_{tk}$ is the portion of samples belonging to class $k$ at node $t$:

$$p_{tk} = \frac{N_{tk}}{N_t}, \qquad \sum_{k=1}^{K} p_{tk} = 1.$$

The proportions of the parent node are related to the proportions of the child nodes:

$$p_{Pk} = \frac{N_{Pk}}{N_P} = \frac{N_{Lk} + N_{Rk}}{N_p} = \frac{N_L p_{Lk} + N_R p_{Rk}}{N_P}.$$

- A node is *pure*, if it only contains samples of one class. It becomes a leaf.
- Not all leaves are pure.
- *Impurity* is a function of class proportions

$$i(t) = \phi(p_{t1}, \ldots, p_{tk}).$$

- Change in impurity $\Delta i$ is the impurity of the parent node minus the weighted average of the impurities of the child nodes:

$$\Delta i = i(P) - \left( \frac{N_L}{N_P} i(L) + \frac{N_R}{N_P} i(R) \right).$$

Properties of impurity:

- It should be zero, if only one class is present. That is $i(t) = 0$, if and only if $p_{tk} = 1$ for some $k$, and zero for all others.
- It should be maximal, when all classes are mixed in equal proportions, i.e. $p_{t1} = \ldots = p_{tK} = 1/K$.
- It should be symmetric, if the classes are re-labeled.

# Gini Diversity Index

The *Gini Diversity Index (gdi)* is

$$i(t) = \sum_{k=1}^{K} p_{tk}(1 - p_{tk}) = 1 - \sum_{k=1}^{K} p_{tk}^2,$$

since $\sum_{k=1}^{K} p_{tk} = 1$. It vanishes, if $p_{tk} = 1$ for some $k$, and zero for all others. The change in impurity is

$$\begin{aligned} \Delta i &= \left(1 - \sum_{k=1}^{K} p_{Pk}^2\right) - \frac{N_L}{N_P}\left(1 - \sum_{k=1}^{K} p_{Lk}^2\right) - \frac{N_R}{N_P}\left(1 - \sum_{k=1}^{K} p_{Rk}^2\right) \\ &= \sum_{k=1}^{K} \frac{N_R}{N_P} p_{Rk}^2 + \frac{N_L}{N_P} p_{Lk}^2 - p_{Pk}^2 \\ &= \frac{N_R}{N_P}\frac{N_L}{N_P} \sum_{k=1}^{K} (p_{Lk} - p_{Rk})^2 . \end{aligned}$$

# Twoing

*Towing* calculates the change in impurity as

$$\Delta i = \frac{N_L}{N_P}\frac{N_R}{N_P}\left(\sum_{k=1}^{K}|p_{Lk} - p_{Rk}|\right)^2.$$

It is commonly used if there are many classes, i.e. $K$ is large.

# Deviance - Cross-Entropy

The *deviance*, also known as *cross-entropy* is

$$i(t) = -\sum_{k=1}^{K} p_{tk} \log p_{tk}.$$

The change in impurity is

$$\Delta i = -\sum_{k=1}^{K} p_{Pk} \log p_{Pk} + \frac{N_L}{N_P} \sum_{k=1}^{K} p_{Lk} \log p_{Lk} + \frac{N_R}{N_P} \sum_{k=1}^{K} p_{Rk} \log p_{Rk}.$$

Using $p_{tk} = N_{tk}/N_t$ and $N_{Pk} = N_{Lk} + N_{Rk}$, this can be rewritten as

$$\Delta i = \frac{1}{N_P} \sum_{k=1}^{K} N_{Lk} \log \frac{p_{Lk}}{p_{Pk}} + N_{Rk} \log \frac{p_{Rk}}{p_{Pk}}.$$

# Node Error

If a node is assigned the class with the largest proportion of training samples at that node, the node error is the fraction of misclassified samples:

$$i(t) = 1 - \max_k p_{tk}.$$

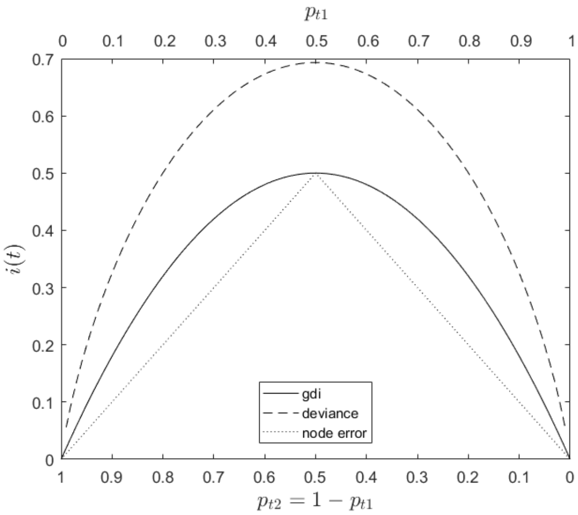
It does not result in a preference to create purer child nodes!

$$\begin{aligned}
\Delta i &= (1 - \max_k p_{Pk}) - \frac{N_L}{N_P}(1 - \max_k p_{Lk}) - \frac{N_R}{N_P}(1 - \max_k p_{Rk}) \\
&= 1 - \max_k \frac{N_{Pk}}{N_P} - \frac{N_L}{N_P} + \frac{N_L}{N_P} \max_k \frac{N_{Lk}}{N_L} - \frac{N_R}{N_P} + \frac{N_R}{N_P} \max_k \frac{N_{Rk}}{N_R} \\
&= \frac{1}{N_P}\left(\max_k N_{Lk} + \max_k N_{Rk} - \max_k N_{Pk}\right),
\end{aligned}$$

where we used $N_L + N_R = N_P$.

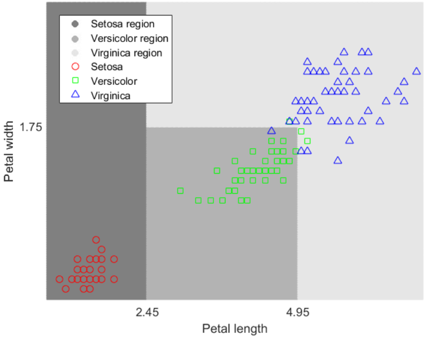
# Node Error

- Suppose there are only two classes and at the parent node they have equal parity, $N_{P1} = N_{P2} = 400$.
- One possible split is $N_{L1} = 100, N_{L2} = 300, N_{R1} = 300, N_{R2} = 100$.
- Another one is $N_{L1} = 200, N_{L2} = 400, N_{R1} = 200, N_{R2} = 0$.
- Both result in $\Delta i = (600 - 400)/800 = 1/4$.
- The second split is preferable, since there the right node is pure, and thus a leaf. For the first split, both child nodes are impure and need to grow branches.

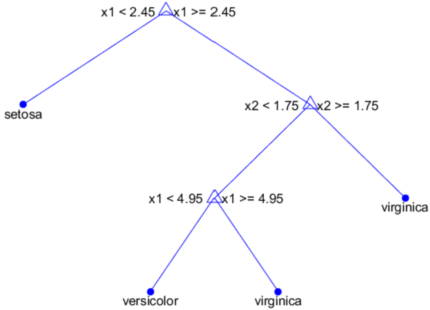


$p_{t1}$
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
$i(t)$
gdi
deviance
node error
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
$p_{t2} = 1 - p_{t1}$

# Decision Trees




(a)




(b)

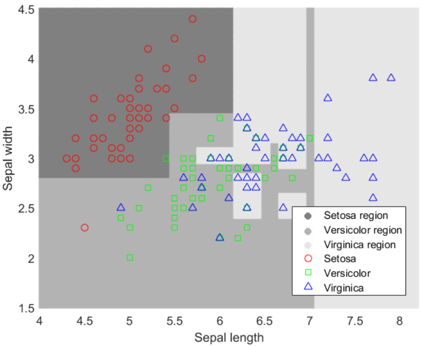
Figure: Decision tree classification.

# Decision Trees

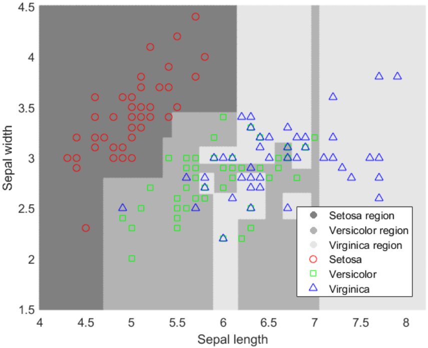
The depth of decision trees is controlled by three parameters:

- the maximum number of allowed branch node splits,
- the minimum number of samples needed in a node which is split,
- and the minimum number of samples per leaf node.

# Decision Trees



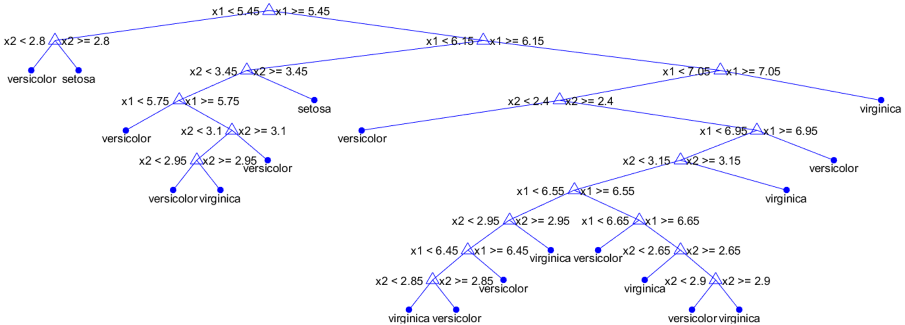

(a) Minimum number per split node set to 10.



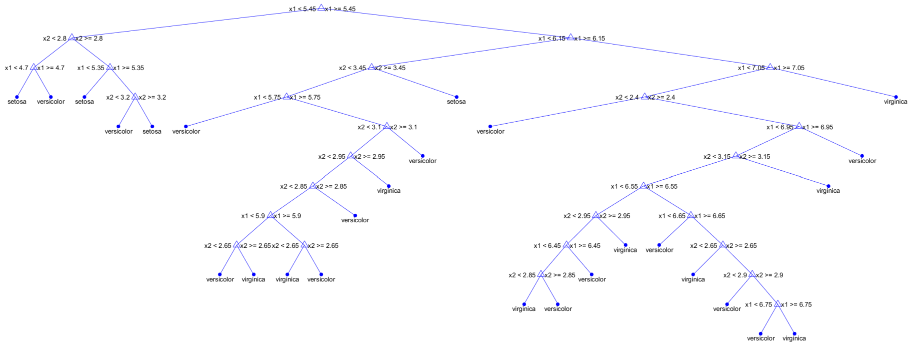

(b) Minimum number per split node set to 5.

Figure: Decision tree classification.

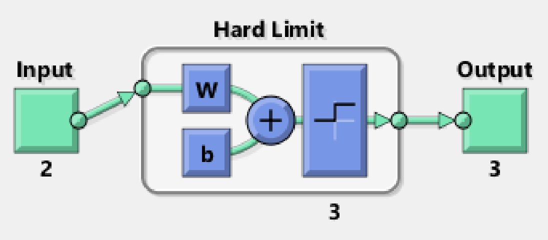
# Decision Trees



# Decision Trees



Recall that the perceptron can be viewed as a single layer *neural network* consisting of input neurons and one output neuron.

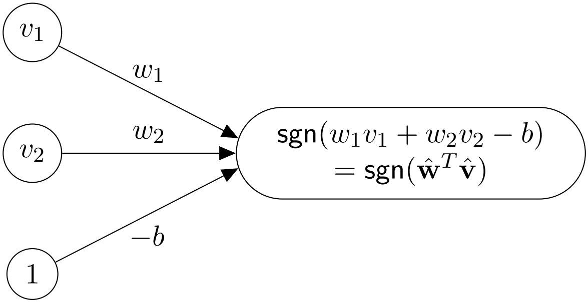


# Neural Networks - Layers

Neural networks are dynamical systems characterized by non-linear, distributed, parallel and local processing. A neural network consists of *neurons* (*nodes* , *units*) and *synapses* connecting the neurons. The neurons are organized into three types of layers:

- *Input*: Each feature is an input neuron.
- *Hidden*: Each Neuron is a (possibly complex) mathematical function creating a predictor.
- *Output*: The neurons gather the predictions and produce the final result.

The synapses not only connect neurons, but also store weights.




The weights are updated by the chosen learning process.

The output neuron

$$\boxed{\begin{array}{c}\mathsf{sgn}(w_1 v_1 + w_2 v_2 - b) \\ = \mathsf{sgn}(\hat{\mathbf{w}}^T \hat{\mathbf{v}})\end{array}}$$

consists of two elements:

- the *propagation function*

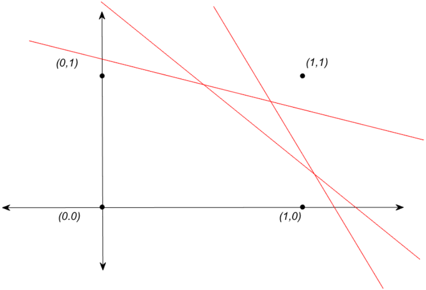
$$f_{prop}(\hat{\mathbf{v}}, \hat{\mathbf{w}}) = \hat{\mathbf{w}}^T \hat{\mathbf{v}}$$

- and the *activation function* $f_{act}(x) = \mathsf{sgn}(x)$.

# Neural Networks - AND

A neural network is capable of implementing the logical AND.

| $v_1$ | $v_2$ | AND |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



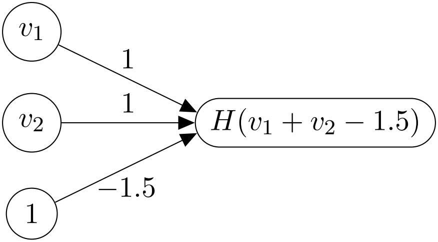

Each line is described by $w_1v_1 + w_2v_2 - b = 0$. For any of the lines, points to the right should result in output $1$.

# Neural Networks - AND

Using the Heaviside step function defined by

$$H(x) = \begin{cases} 0 & \text{if} \quad x \leq 0 \\ 1 & \text{if} \quad x > 0 \end{cases},$$

the resulting neural network might be:

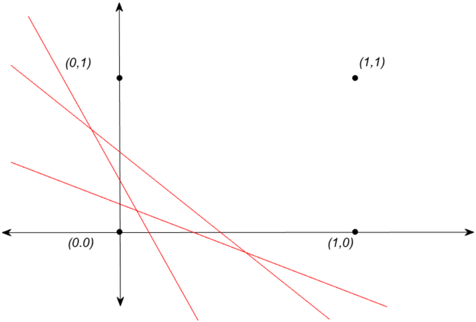



Note, the learning process might have arrived at different choices for $w_1$, $w_2$ and $b$.

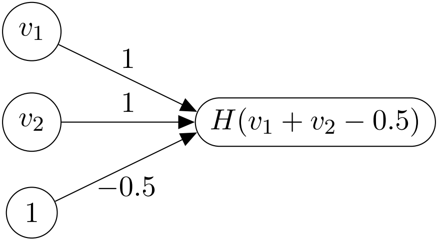
# Neural Networks - OR

A neural network is capable of implementing the logical OR.

| $v_1$ | $v_2$ | OR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



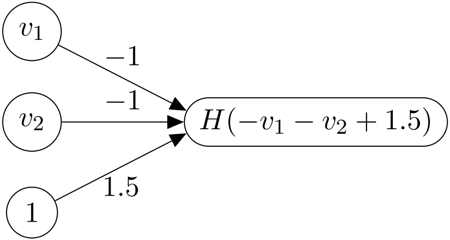
Using the Heaviside step function again the resultant neural network might be




Note, the learning process might have arrived at different choices for $w_1$, $w_2$ and $b$.

A neural network is capable of implementing the logical AND NOT $=$ NAND, for example:

| $v_1$ | $v_2$ | NAND |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

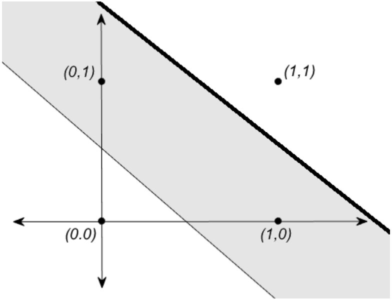



Note, the learning process might have arrived at different choices for $w_1$, $w_2$ and $b$.

# Neural Networks - XOR

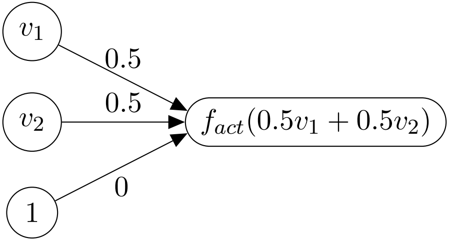
A simple neural network is **not** capable of implementing the exclusive OR = XOR (one or the other, but not both).

| $v_1$ | $v_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



An activation function with two steps is possible, for example

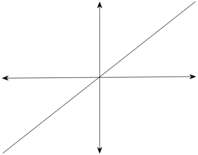
$$f_{act}(x) = \begin{cases} 0 & \text{if} \quad x \leq 0.25 \\ 1 & \text{if} \quad 0.25 < x < 0.75 \\ 0 & \text{if} \quad x \geq 0.75 \end{cases}$$



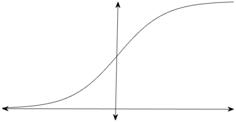

Note, the learning process might have arrived at different choices for $w_1$, $w_2$ and $b$.

## Neural Networks - Activation Functions

- Linear $f_{act}(x) = x$

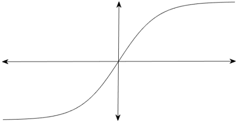


- Sigmoid $f_{act}(x) = \dfrac{1}{1 + e^{-a(x-c)}}$

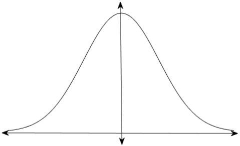


- Hyperbolic tangent

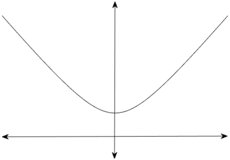
$$f_{act}(x) = \tanh(x)$$



- Gaussian $f_{act}(x) = \exp\left(-\frac{(x-c)^2}{2a^2}\right)$

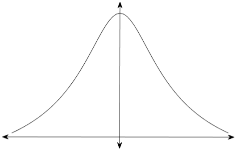


- Multiquadratics $f_{act}(x) = \sqrt{(x-c)^2 + a^2}$



- Inverse multiquadratics

$$f_{act}(x) = \frac{1}{\sqrt{(x-c)^2 + a^2}}$$



# Softmax

*Softmax activation function*, aka *normalized exponential function*

- Takes into account the result of the propagation functions of other neurons.
- Let $z_j$ be result of the propagation function in the $j^{\text{th}}$ neuron: $z_j = \hat{\mathbf{w}}_j^T \hat{\mathbf{v}}$.
- If the number of neurons is $K$, then the softmax function maps the $K$-dimensional vector $\mathbf{z} = (z_1, \ldots, z_K)^T$ to a $K$-dimensional vector $\boldsymbol{\sigma}(\mathbf{z})$:

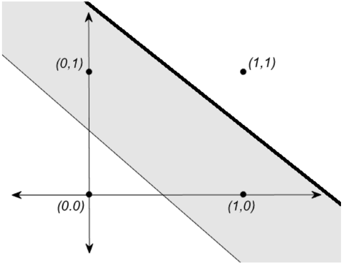
$$\boldsymbol{\sigma}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}.$$

- Obviously, the elements of $\boldsymbol{\sigma}(\mathbf{z})$ sum to $1$.

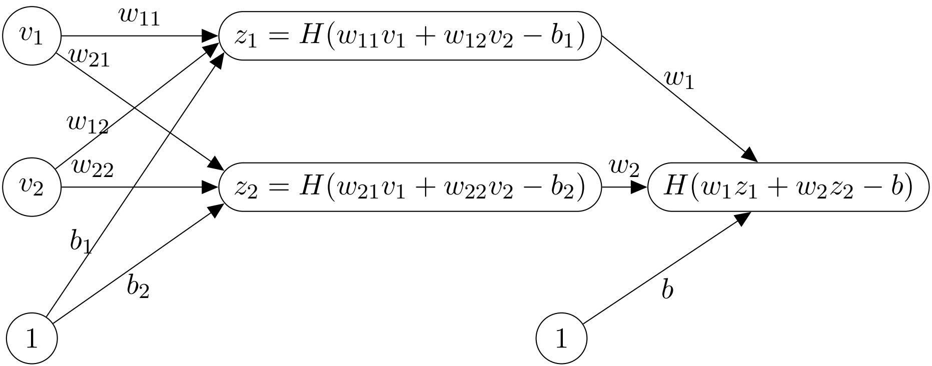
# Neural Networks - XOR

- However, the construction of activation function is not in the spirit of machine learning, where as many tasks as possible shall be completed by the machine.
- XOR can also be implemented by introducing a *hidden* layer.

# Neural Networks - XOR



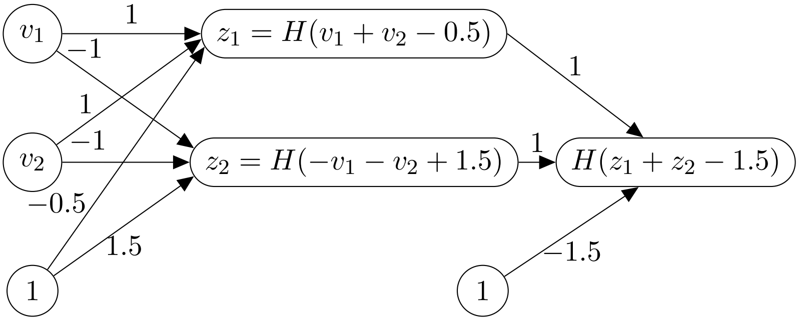

Two lines are necessary. Points on the right of the thin line and on the left of the bold line shall output $1$.



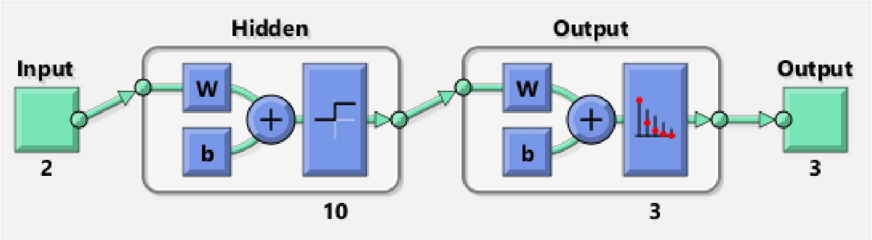
$v_1$
$v_2$
1
1
$w_{11}$
$w_{21}$
$w_{12}$
$w_{22}$
$b_1$
$b_2$
$z_1 = H(w_{11}v_1 + w_{12}v_2 - b_1)$
$z_2 = H(w_{21}v_1 + w_{22}v_2 - b_2)$
$w_1$
$w_2$
$b$
$H(w_1 z_1 + w_2 z_2 - b)$

# Neural Networks - XOR

For example:

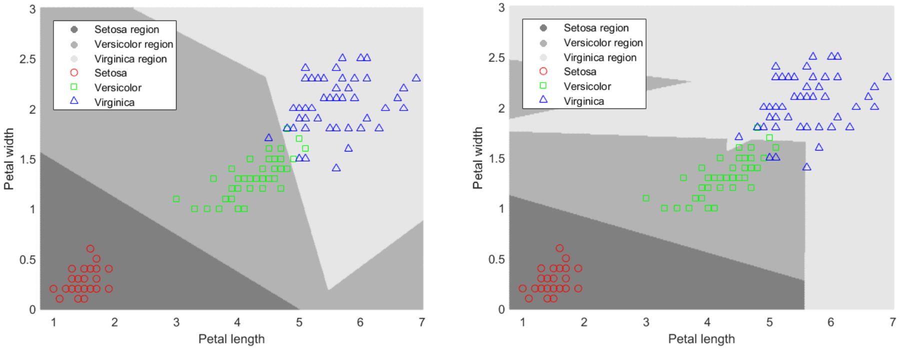



$z_1$ represents an OR (on the right of the thin line), while $z_2$ represents NAND (on the left of the bold line). The output neuron combines these with an AND, since both have to hold at the same time.




Neural network with 2 input neurons, 10 neurons in the hidden layer using the heaviside step function, and 3 output neurons using the softmax function.

# Neural Networks




(a) All samples used for training.

(b) 70% of samples used for training, 15% used for validation, 15% used for testing.

Figure: Neural network classification with one hidden layer containing 10 neurons.

# Neural Networks

A neural network is sensitive to

- the number of layers,
- the connection pattern,
- the initialization of weights and
- the activation functions and their parameters.

# Boosting

Definition:

- Weak learner: slightly better than random guessing.
- Strong learner: right classification most of the time.
- Can a set of weak learners create a single strong learner?

# Boosting

Returning to binary classification:

- We assume the number of samples in each class is the same.
- The sign of the output of the classifier $\mathcal{C}$ determines the class.
- The absolute value of the output gives the confidence in the prediction.

# Boosting

- The total error $E$ of $\mathcal{C}$ is the sum of exponential error at each training sample
$$E(\mathcal{C}) = \sum_{i=1}^{N} \exp\left(-c_i \mathcal{C}(\mathbf{v}_i)\right),$$
where $c_i = \pm 1$ is the class label of $\mathbf{v}_i$.
- If $\mathcal{C}$ predicts the class label of $\mathbf{v}_i$ **correctly** with great confidence, then $\exp\left(-c_i \mathcal{C}(\mathbf{v}_i)\right)$ will contribute very little to the total error.
- If $\mathcal{C}$ predicts the class label of $\mathbf{v}_i$ **incorrectly** with great confidence, then $\exp\left(-c_i \mathcal{C}(\mathbf{v}_i)\right)$ will contribute a lot to the total error.
- Aim: A classifier which predicts correctly with great confidence and has little confidence in the prediction when making errors.

AdaBoost - *Adaptive Boosting*

- Iteratively generates a series of classifiers $\mathcal{C}_m$.
- Each iteration improves the classifier by concentrating on the misclassified elements.
- Each classifier is stronger than its predecessor.
- Each classifier is a linear combination of weak classifiers $k_j$ (returning $\pm 1$) with positive coefficients.
- Each coefficient gives the confidence in the weak classifier.

AdaBoost - Adaptive Boosting

- $\mathcal{C}_1$ is initialized to the weak classifier $k_1(\mathbf{v})$ which misclassifies the least number of training examples.
- Its coefficient $\alpha_1$ is chosen to minimize

$$\begin{aligned} E(\mathcal{C}_1) &= \sum_{i=1}^{N} \exp\left(-c_i \alpha_1 k_1(\mathbf{v}_i)\right) \\ &= \sum_{c_i \neq k_1(\mathbf{v}_i)} \exp\left(\alpha_1\right) + \sum_{c_i = k_1(\mathbf{x}_i)} \exp\left(-\alpha_1\right). \end{aligned}$$

- Differentiating with respect to $\alpha_1$

$$\frac{dE(\mathcal{C}_1)}{d\alpha_1} = \sum_{c_i \neq k_1(\mathbf{v}_i)} \exp\left(\alpha_1\right) - \sum_{c_i = k_1(\mathbf{v}_i)} \exp\left(-\alpha_1\right).$$

# Adaptive Boosting

- Let $N$ be the number of samples and $N_C$ be number of correctly classified samples.
- We have a minimum for

$$\alpha_1 = \frac{1}{2} \ln \frac{N_C}{N - N_C}.$$

- If $N_C = N/2$, then $\alpha_1 = 0$, that is no confidence in the classification, since it is equal to random guessing.
- The larger $N_C$, the higher the confidence.

# Adaptive Boosting

- In the $m$-th iteration we generate $\mathcal{C}_m = \mathcal{C}_{m-1} + \alpha_m k_m$.
- The total error is

$$
\begin{aligned}
E(\mathcal{C}_m) &= \sum_{i=1}^{N} \exp\left(-c_i \mathcal{C}_{m-1}(\mathbf{v}_i) - c_i \alpha_m k_m(\mathbf{v}_i)\right) \\
&= \sum_{i=1}^{N} \exp\left(-c_i \mathcal{C}_{m-1}(\mathbf{v}_i)\right) \exp\left(-c_i \alpha_m k_m(\mathbf{v}_i)\right) \\
&= \exp\left(-\alpha_m\right) \sum_{c_i = k_m(\mathbf{v}_i)} w_{i,m} + \exp\left(\alpha_m\right) \sum_{c_i \neq k_m(\mathbf{v}_i)} w_{i,m},
\end{aligned}
$$

where we defined the weights $w_{i,m} = \exp\left(-c_i \mathcal{C}_{m-1}(\mathbf{v}_i)\right)$.

# Adaptive Boosting

- The error can be rewritten as

$$
\begin{aligned}
E(\mathcal{C}_m) \quad &= \quad \exp(-\alpha_m) \sum_{i=1}^{N} w_{i,m} \\
& \quad + (\exp(\alpha_m) - \exp(-\alpha_m)) \sum_{c_i \neq k_m(\mathbf{v}_i)} w_{i,m}.
\end{aligned}
$$

- Thus the weak classifier where the sum of weights over the misclassified elements is smallest should be chosen.
- That is the classifier which classifies most samples with large weights correctly.
- $\alpha_m$ is then chosen to minimize $E$, that is

$$
\alpha_m = \frac{1}{2} \ln \frac{\sum_{c_i = k_m(\mathbf{x}_i)} w_{i,m}}{\sum_{c_i \neq k_m(\mathbf{x}_i)} w_{i,m}}.
$$

# Cascade

So far we have not considered the cost of acquiring features. These cost could be:

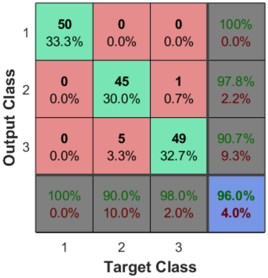
- computational (simple to computationally intensive algorithms)
- financial (cheap to expensive diagnostics)
- human (simple medical tests to invasive procedures)

Each stage of a *cascade* of classifiers uses features with increasing predictive power and increasing cost, for example medical diagnosis.

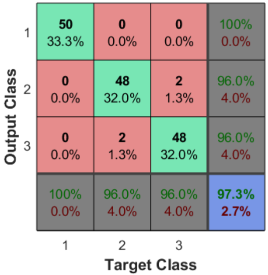
# Confusion matrix

- Let class $C_0$ be the *negatives* and class $C_1$ be the *positives*.
- N: number of samples in $C_0$, P: number of samples in $C_1$.
- *True negatives* TN: number of samples of $C_0$ correctly classified.
- *False positives* FP: the number of samples of $C_0$ misclassified.
- *True positives* TP: number of samples of $C_1$ correctly classified.
- *False negatives* FN: number of samples of $C_1$ misclassified.
- *Confusion matrix*:

$$\left( \begin{array}{cc} \mathrm{TN} & \mathrm{FN} \\ \mathrm{FP} & \mathrm{TP} \end{array} \right).$$




(a) All samples used for training.




(b) $70\%$ of samples used for training, $15\%$ used for validation, $15\%$ used for testing.

Figure: Confusion matrices.

# Cascade

- *Sensitivity*, aka *recall*, *true positive rate* and *probability of detection*: fraction of positive samples correctly classified: $\mathrm{TPR} = \dfrac{\mathrm{TP}}{\mathrm{P}}$.
- *Specificity* or *true negative rate*: fraction of negative samples correctly identified: $\mathrm{TNR} = \dfrac{\mathrm{TN}}{\mathrm{N}}$.
- *False negative rate* or *miss rate*: $\mathrm{FNR} = \dfrac{\mathrm{FN}}{\mathrm{P}} = 1 - \mathrm{TPR}$.
- *Fall-out* or *false positive rate*: $\mathrm{FPR} = \dfrac{\mathrm{FP}}{\mathrm{N}} = 1 - \mathrm{TNR}$.

A perfect classifier would be $100\%$ sensitive (all positives are correctly identified) and $100\%$ specific (no negatives are incorrectly classified).

# Cascade

- The early stages have high sensitivity.
- The later stages have high specificity.
- An $S$ stage cascade has classifiers $C_1, \ldots, C_S$ where the $M$-dimensional feature vector $\mathbf{x}$ is divided into $S$ distinct sets $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_S)$.
- Samples classified as negative at a stage are not considered in the following stages, thus decreasing the overall computational cost.