$k-$NN and Linear Classifiers, Learning

Tomáš Svoboda and Matěj Hoffmann
thanks to Daniel Novák and Filip Železný, Ondřej Drbohlav

Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

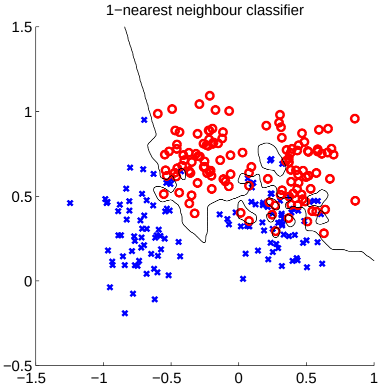
May 20, 2020

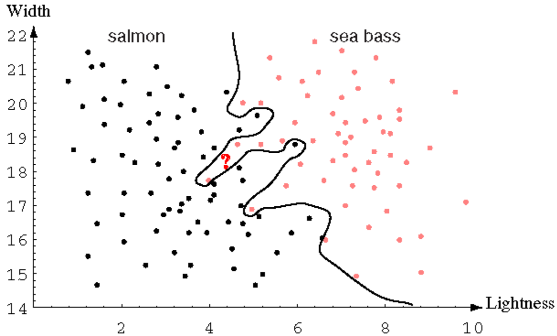
Notes

K-Nearest neighbors classification

For a query $\vec{x}$:

- Find K nearest $\vec{x}$ from the tranining (labeled) data.
- Classify to the class with the most exemplars in the set above.





Notes

Some properties:

- A *nonparametric* method – does not assume anything about the distribution (that it is Gaussian etc.)
- Can be used for classification or regression. Here: classification.
- Training: Only store feature vectors and their labels.
- Very simple and suboptimal. With unlimited nr. prototypes, error never worse than twice the Bayes rate (optimum).
- *instance-based* or *lazy* learning – function only approximated locally; computation only during inference.
- Limitations
  - Curse of dimensionality - for every additional dimension, one needs exponentially more points to cover the space.
  - Comp. complexity - has to look through all the samples all the time. Some speed-up is possible. E.g., storing data in a K-d tree.
  - Noise. Missclassified examples will remain in the database....

$K-$ Nearest Neighbor and Bayes $j^* = \text{argmax}_j \, P(s_j|\vec{x})$

Assume data:

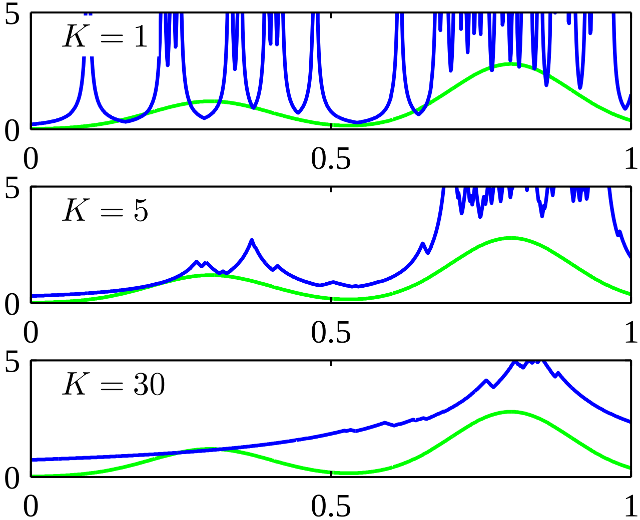
- N points $\vec{x}$ in total.
- N_j points in s_j class. Hence, $\sum_j N_j = N$.

We want classify $\vec{x}$. We draw a sphere centered at $\vec{x}$ containing K points irrespective of class. V is the volume of this sphere. $P(s_j|\vec{x}) = ?$

$$P(s_j|\vec{x}) = \frac{P(\vec{x}|s_j)P(s_j)}{P(\vec{x})}$$

$$\begin{aligned}
P(s_j) &= \frac{N_j}{N} \\
P(\vec{x}) &= \frac{K}{NV} \\
P(\vec{x}|s_j) &= \frac{K_j}{N_j V} \\
P(s_j|\vec{x}) &= \frac{P(\vec{x}|s_j)P(s_j)}{P(\vec{x})} = \frac{K_j}{K}
\end{aligned}$$

Notes



A $K-$NN classifier can be understood as a non-parametric density estimator. (Figure from [1])

$K-$ Nearest Neighbor and Bayes $j^* = \text{argmax}_j \, P(s_j|\vec{x})$

Assume data:

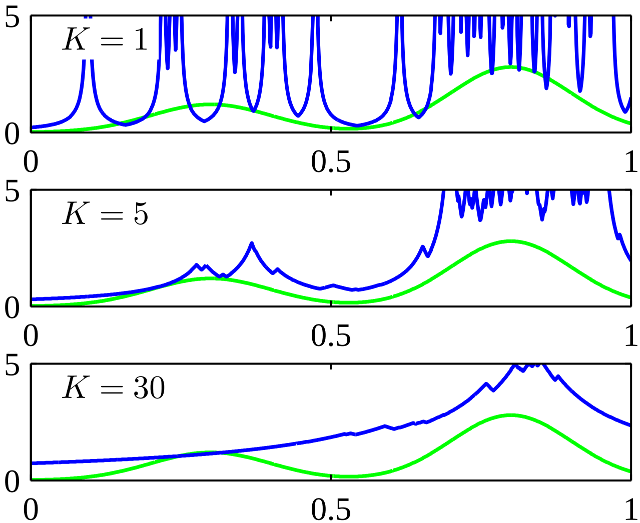
- N points $\vec{x}$ in total.
- N_j points in s_j class. Hence, $\sum_j N_j = N$.

We want classify $\vec{x}$. We draw a sphere centered at $\vec{x}$ containing K points irrespective of class. V is the volume of this sphere. $P(s_j|\vec{x}) = ?$

$$P(s_j|\vec{x}) = \frac{P(\vec{x}|s_j)P(s_j)}{P(\vec{x})}$$

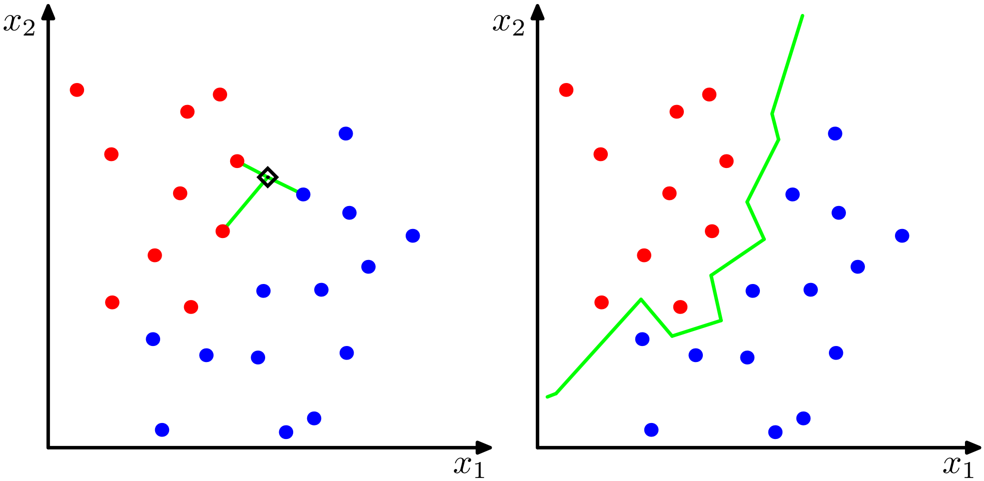
$$\begin{aligned}
P(s_j) &= \frac{N_j}{N} \\
P(\vec{x}) &= \frac{K}{NV} \\
P(\vec{x}|s_j) &= \frac{K_j}{N_j V} \\
P(s_j|\vec{x}) &= \frac{P(\vec{x}|s_j)P(s_j)}{P(\vec{x})} = \frac{K_j}{K}
\end{aligned}$$

Notes



A $K-$NN classifier can be understood as a non-parametric density estimator. (Figure from [1])

NN classification example

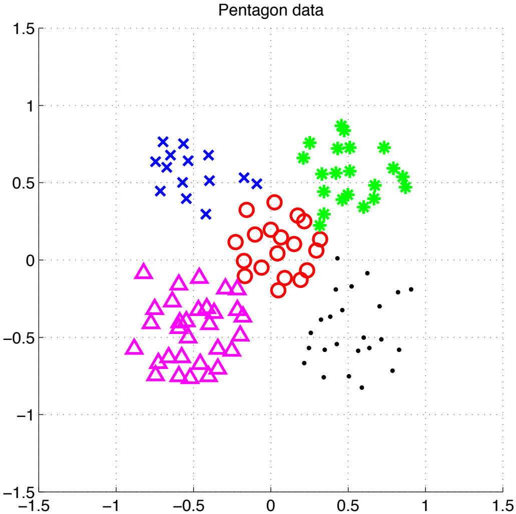


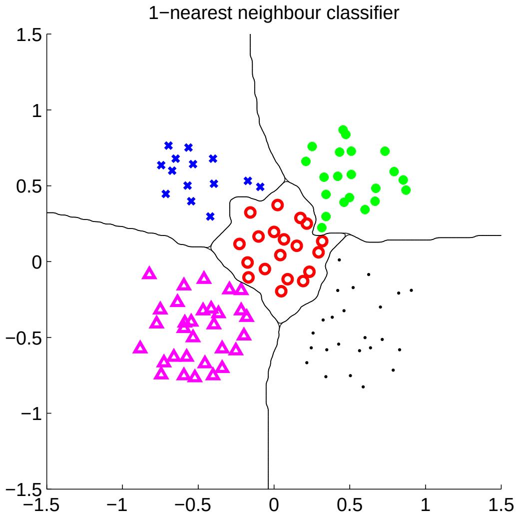
(a) (b) [1]

[1] Figs from [1]

Notes

NN classification example





Notes

Fast on "learning", very slow on decision.

There are ways for speeding it up, search for NN editing - making training data sparser, keeping only representative points.

What is *nearest*? Metrics for NN classification ...

A function D which is: nonnegative, reflexive, symmetrical, satisfying triangle inequality:

$D(\vec{a}, \vec{b}) \geq 0$
$D(\vec{a}, \vec{b}) = 0$ iff $\vec{a} = \vec{b}$
$D(\vec{a}, \vec{b}) = D(\vec{b}, \vec{a})$
$D(\vec{a}, \vec{b}) + D(\vec{b}, \vec{c}) \geq D(\vec{a}, \vec{c})$

Notes

When taking $\vec{x}$ as all the intenties, "5" shifted 3 pixels left is farther from its etalon thant to etalon of "8". One could consider preprocessing:

1. shift query image to all possible positions and compute min distances
2. take the min(min(distance))
3. perform NN classification

Costly ...

What is *nearest*? Metrics for NN classification ...

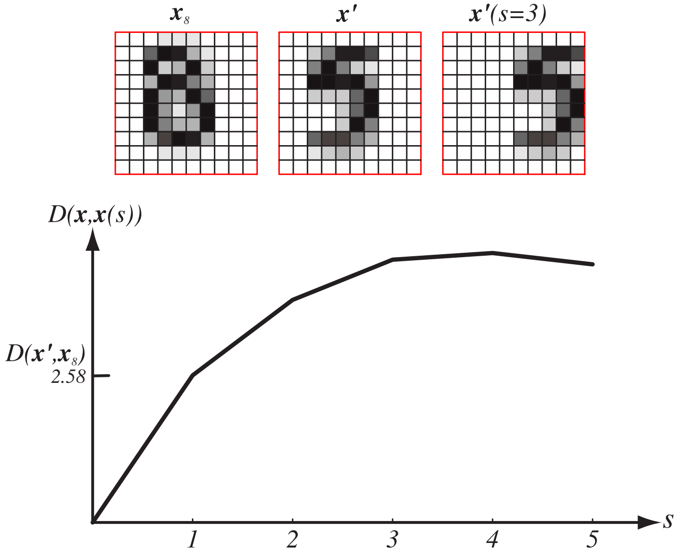
A function D which is: nonnegative, reflexive, symmetrical, satisfying triangle inequality:

$D(\vec{a}, \vec{b}) \geq 0$

$D(\vec{a}, \vec{b}) = 0$ iff $\vec{a} = \vec{b}$

$D(\vec{a}, \vec{b}) = D(\vec{b}, \vec{a})$

$D(\vec{a}, \vec{b}) + D(\vec{b}, \vec{c}) \geq D(\vec{a}, \vec{c})$



Invariance to geometrical transformations? (figure from [2])

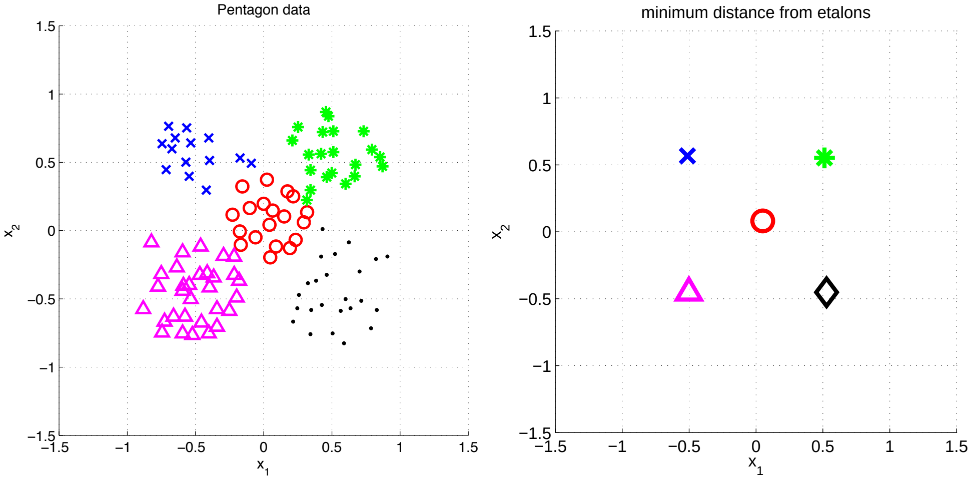
Notes

When taking $\vec{x}$ as all the intenties, "5" shifted 3 pixels left is farther from its etalon thant to etalon of "8". One could consider preprocessing:

1. shift query image to all possible positions and compute min distances
2. take the min(min(distance))
3. perform NN classification

Costly ...

Etalon based classification

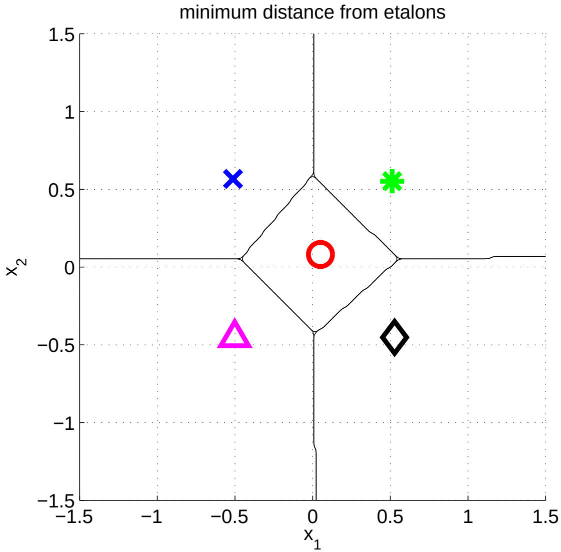


Represent $\vec{x}$ by etalon , $\vec{e}_s$ per each class $s \in S$

Notes

Separate etalons

$$s^* = \arg\min_{s \in S} \|\vec{x} - \vec{e}_s\|^2$$



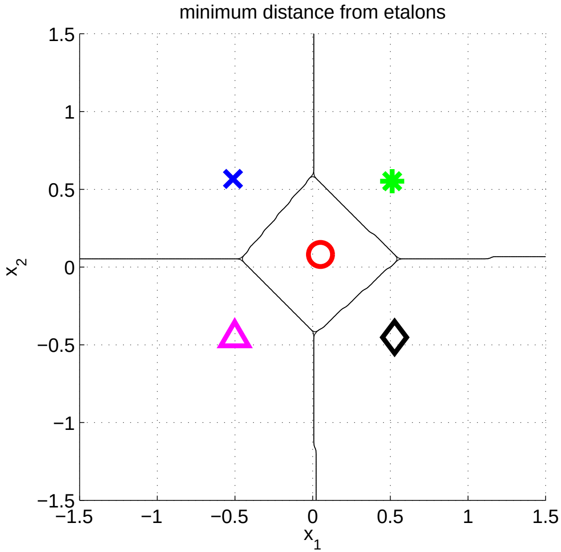
Notes

What etalons?

If $\mathcal{N}(\vec{x}|\vec{\mu},\Sigma)$; all classes same covariance matrices, then

$$\vec{e}_s \stackrel{\text{def}}{=} \vec{\mu}_s = \frac{1}{|\mathcal{X}^s|}\sum_{i\in\mathcal{X}^s}\vec{x}_i^s$$

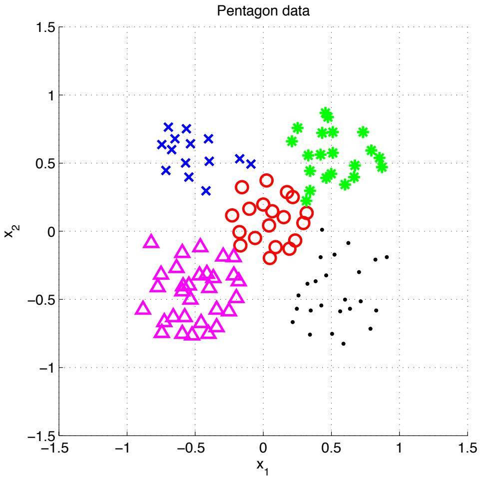
and separating hyperplanes halve distances between pairs.

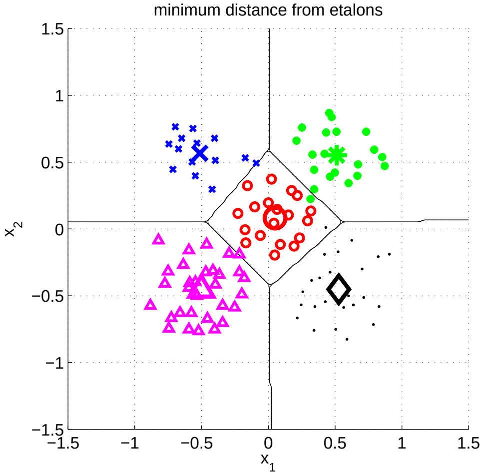


Notes

$$\mathcal{N}(\vec{x}|\vec{\mu},\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\{-\frac{1}{2}(\vec{x}-\vec{\mu})^\top\Sigma^{-1}(\vec{x}-\vec{\mu})\}$$

Etalon based classification, $\vec{e}_s = \vec{\mu}_s$

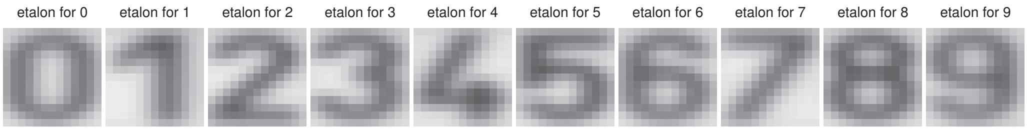




Notes

Some wrongly classified samples. We like the simple idea. Are there better etalons? How to find them?

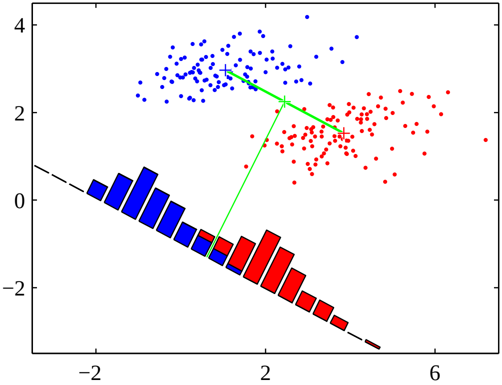
Digit recognition - etalons $\vec{e}_s = \vec{\mu}_s$



Figures from [5]

Notes

Better etalons – Fischer linear discriminant



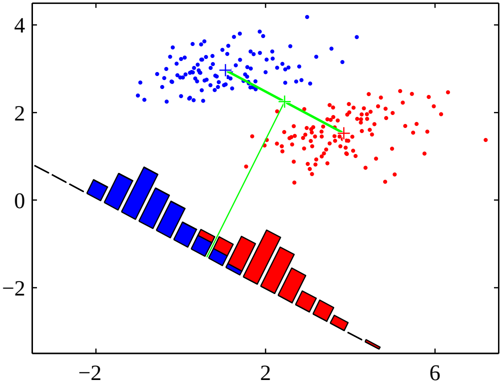
- Dimensionality reduction
- Maximize distance between means, ...
- ...and minimize within class variance. (minimize overlap)

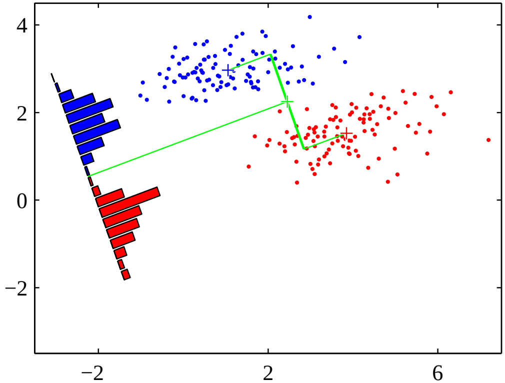
Figures from [1]

Notes

Searching for a projection of the data to minimize intra-class variance and maximize inter-class variance.

Better etalons – Fischer linear discriminant





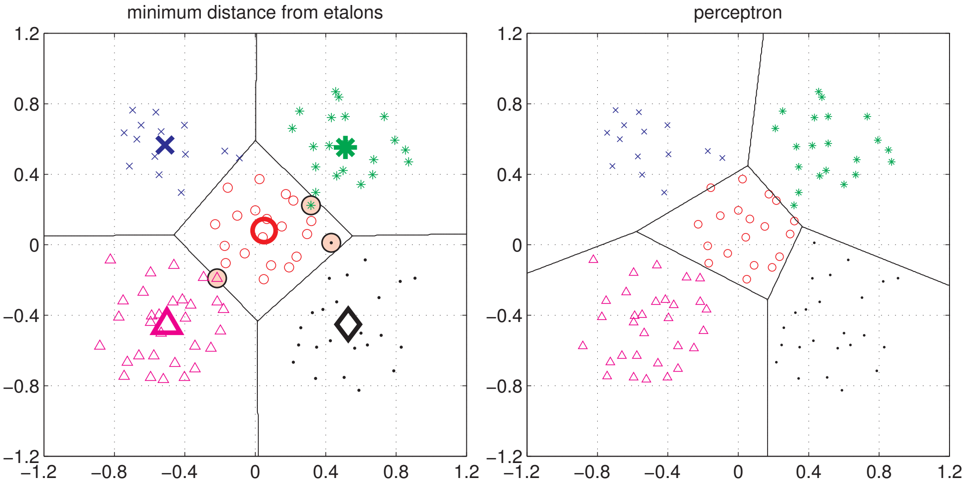
- Dimensionality reduction
- Maximize distance between means, . . .
- . . . and minimize within class variance. (minimize overlap)

Figures from [1]

Notes

Searching for a projection of the data to minimize intra-class variance and maximize inter-class variance.

Better etalons?



Figures from [5]

Notes

This is just to show that there is an etalon classifier that make no mistake on the data. But how to find the best etalons?

Etalon classifier – Linear classifier

$$
\begin{aligned}
s^* &= \arg\min_{s\in S} \|\vec{x} - \vec{e}_s\|^2 = \arg\min_{s\in S}(\vec{x}^\top \vec{x} - 2\,\vec{e}_s^\top \vec{x} + \vec{e}_s^\top \vec{e}_s) = \\
&= \arg\min_{s\in S}\left(\vec{x}^\top \vec{x} - 2\left(\vec{e}_s^\top \vec{x} - \frac{1}{2}(\vec{e}_s^\top \vec{e}_s)\right)\right) = \\
&= \arg\min_{s\in S}\left(\vec{x}^\top \vec{x} - 2\left(\vec{e}_s^\top \vec{x} + b_s\right)\right) = \\
&= \boxed{\arg\max_{s\in S}(\vec{e}_s^\top \vec{x} + b_s)} = \arg\max_{s\in S} g_s(\vec{x}). \qquad b_s = -\frac{1}{2}\vec{e}_s^\top \vec{e}_s
\end{aligned}
$$

Linear function (plus offset)

$$g_s(\mathbf{x}) = \mathbf{w}_s^\top \mathbf{x} + w_{s0}$$

Notes

The result is a *linear discriminant function* – hence etalon classifier is a linear classifier.
We classify into the class with highest value of the discriminant function.
$\mathbf{w}_s$ is a generalized etalon. How do we find it? Such that it is better than just the mean of the class members in the training set.

Learning and decision

Learning stage - learning models/function/parameters from data.

Decision stage - decide about a query $\vec{x}$.

What to learn?

- Generative model : Learn $P(\vec{x}, s)$. Decide by computing $P(s|\vec{x})$.
- Discriminative model : Learn $P(s|\vec{x})$
- Discriminant function : Learn $g(\vec{x})$ which maps $\vec{x}$ directly into class labels.

Notes

Generative models because by sampling from them it is possible to generate synthetic data points $\vec{x}$.
For the discriminative model one can consider, e.g. logistic function:

$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$

(1) Linear discriminant function - two class case

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

Decide s_1 if $g(\mathbf{x}) > 0$ and s_2 if $g(\mathbf{x}) < 0$

Figure from [2]

Notes

$g(\mathbf{x}) = 0$ is the *separating hyperplane*. Its dimension is one less that that of the input space – for 2D space, it is a line. (This is a bit counterintuitive - "hyper" normally means above, more...)

What is the geometric meaning of the weight vector $\mathbf{w}$?

(1) Linear discriminant function - two class case

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

Decide s_1 if $g(\mathbf{x}) > 0$ and s_2 if $g(\mathbf{x}) < 0$

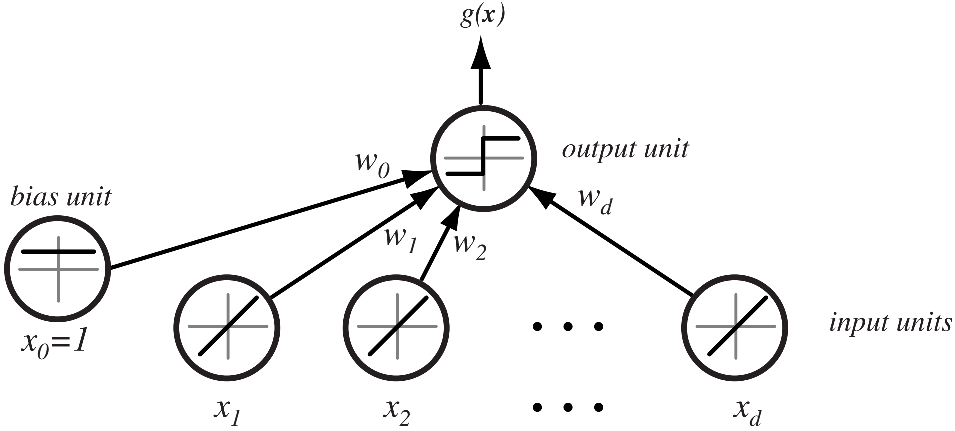


Figure from [2]

Notes

$g(\mathbf{x}) = 0$ is the *separating hyperplane*. Its dimension is one less that that of the input space – for 2D space, it is a line. (This is a bit counterintuitive - “hyper” normally means above, more...)

What is the geometric meaning of the weight vector $\mathbf{w}$?

Separating hyperplane

$$\mathbf{w}^\top \mathbf{x}_1 + w_0 = \mathbf{w}^\top \mathbf{x}_2 + w_0$$
$$\mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2) = 0$$

$g(\mathbf{x})$ gives an algebraic measure of the distance from $\mathbf{x}$ to the hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

as $g(\mathbf{x}_p) = 0$,
and $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$, then:

$$g(\mathbf{x}) = r\|\mathbf{w}\|$$

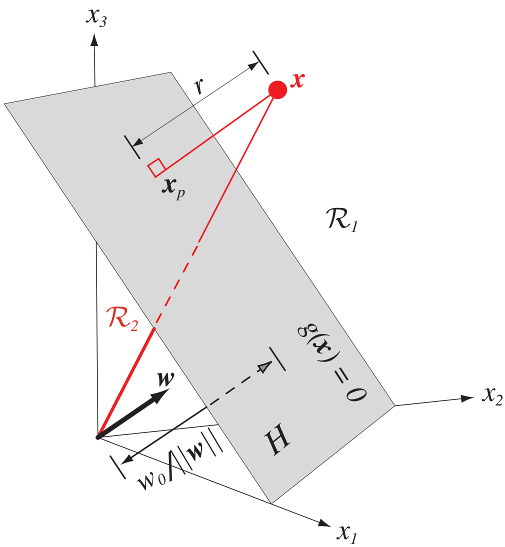


Figure from [2]

Notes

(any) vector $(\mathbf{x}_1 - \mathbf{x}_2)$ lies on the separating hyperplane, $\mathbf{w}$ is perpendicular to it
Summary: A linear discriminant function divides the feature space by a hyperplane decision surface.

- The orientation of the surface is detemined by the normal vector $\mathbf{w}$.
- The location of the surface is determined by the bias term w_0.

Separating hyperplane

$$\mathbf{w}^\top \mathbf{x}_1 + w_0 = \mathbf{w}^\top \mathbf{x}_2 + w_0$$
$$\mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2) = 0$$

$g(\mathbf{x})$ gives an algebraic measure of the distance from $\mathbf{x}$ to the hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

as $g(\mathbf{x}_p) = 0$,
and $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$, then:

$$g(\mathbf{x}) = r\|\mathbf{w}\|$$

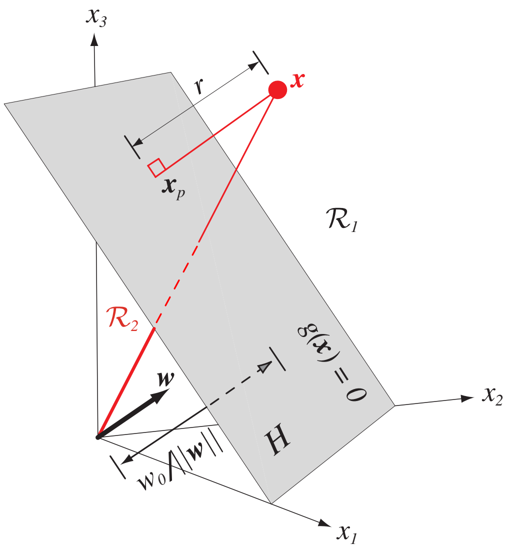


Figure from [2]

Notes

(any) vector $(\mathbf{x}_1 - \mathbf{x}_2)$ lies on the separating hyperplane, $\mathbf{w}$ is perpendicular to it
Summary: A linear discriminant function divides the feature space by a hyperplane decision surface.

- The orientation of the surface is detemined by the normal vector $\mathbf{w}$.
- The location of the surface is determined by the bias term w_0.

Separating hyperplane from g_1 and g_2

$$\begin{aligned} g_1(\vec{x}) &= \vec{\mu}_1^\top \vec{x} - \frac{1}{2}\vec{\mu}_1^\top \vec{\mu}_1 \\ g_2(\vec{x}) &= \vec{\mu}_2^\top \vec{x} - \frac{1}{2}\vec{\mu}_2^\top \vec{\mu}_2 \end{aligned}$$

Separating hyperplane:

$$g_1(\vec{x}) = g_2(\vec{x})$$

$$(\vec{\mu}_1 - \vec{\mu}_2)^\top \vec{x} = \frac{1}{2}(\vec{\mu}_1^\top \vec{\mu}_1 - \vec{\mu}_2^\top \vec{\mu}_2)$$

Notes

Think about case where $\|\vec{\mu}_1\| = \|\vec{\mu}_2\|$ and reason about simplified equation of the separating hyperplane.

Two classes set-up

$|S| = 2$, i.e. two states (typically also classes)

$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\boxed{\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix},\ \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}}$$

for all $\mathbf{x}'$

$$\mathbf{w}'^\top \mathbf{x}' > 0$$

drop the dashes to avoid notation clutter.

Notes

There are two steps here:

1. Transformation to homogenous notation with augmented feature vector and augmented weight vector.
2. "Normalization" that simplifies treatment of the two-class case: labels can be ignored. Just look for a weight vector $\mathbf{w}$ such that $\mathbf{w}^\top \mathbf{x} > 0$

It means, the sign of $\mathbf{x}$ depends on the class it belongs to! Keep in mind.

Two classes set-up

$|S| = 2$, i.e. two states (typically also classes)

$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\boxed{\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix},\ \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}}$$

for all $\mathbf{x}'$

$$\mathbf{w}'^\top \mathbf{x}' > 0$$

drop the dashes to avoid notation clutter.

Notes

There are two steps here:

1. Transformation to homogenous notation with augmented feature vector and augmented weight vector.
2. "Normalization" that simplifies treatment of the two-class case: labels can be ignored. Just look for a weight vector $\mathbf{w}$ such that $\mathbf{w}^\top \mathbf{x} > 0$

It means, the sign of $\mathbf{x}$ depends on the class it belongs to! Keep in mind.

Two classes set-up

$|S| = 2$, i.e. two states (typically also classes)

$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top\mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top\mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\boxed{\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix},\ \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}}$$

for all $\mathbf{x}'$

$$\mathbf{w}'^\top\mathbf{x}' > 0$$

drop the dashes to avoid notation clutter.

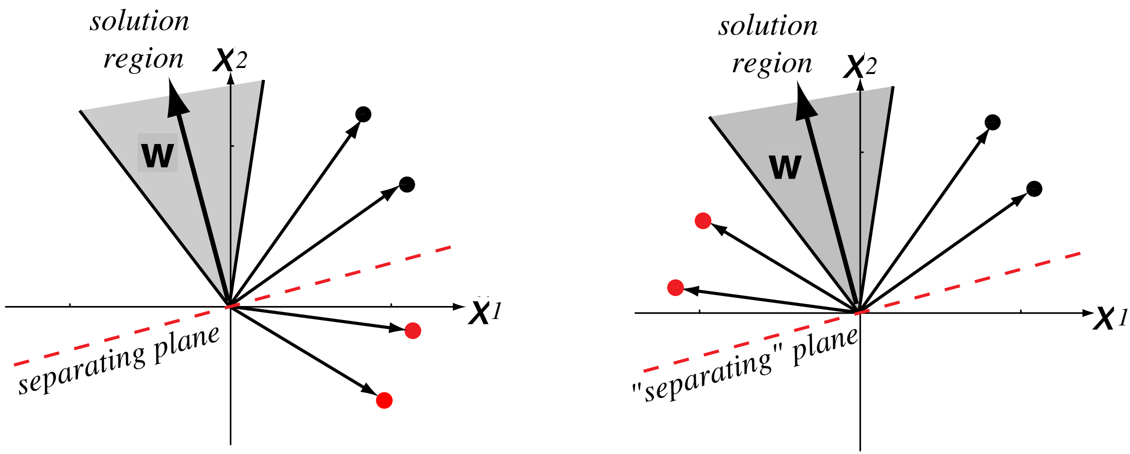
Notes

There are two steps here:

1. Transformation to homogenous notation with augmented feature vector and augmented weight vector.
2. "Normalization" that simplifies treatment of the two-class case: labels can be ignored. Just look for a weight vector $\mathbf{w}$ such that $\mathbf{w}^\top\mathbf{x} > 0$

It means, the sign of $\mathbf{x}$ depends on the class it belongs to! Keep in mind.

Solution (graphically)



Fou

training samples. Left: orginal, Right: sign corrected

Figure from [2] (notation changed)

Notes

Four training samples (black for class/category w_1, red for w_2). Left: Raw data Right: "Normalized data". Class w_2 member replaced by their negatives... Simplifies the situation: labels can be ignored. Just look for a weight vector $\mathbf{w}$ such that $\mathbf{w}^\top \mathbf{x} > 0$
Before: defining the linear discriminant function.
Now: How can we obtain it from (labeled) data?

Learning $\mathbf{w}$, gradient descent

A criterion to be minimized $J(\mathbf{w})$; assume to be known

Initialize $\mathbf{w}$, threshold θ, learning rate α
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha(k)\nabla J(\mathbf{w})$
until $|\alpha(k)\nabla J(\mathbf{w})| < \theta$
return $\mathbf{w}$

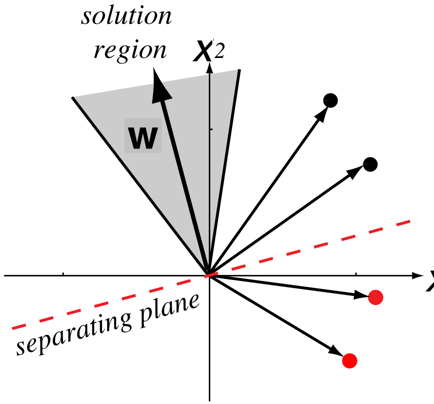
Notes

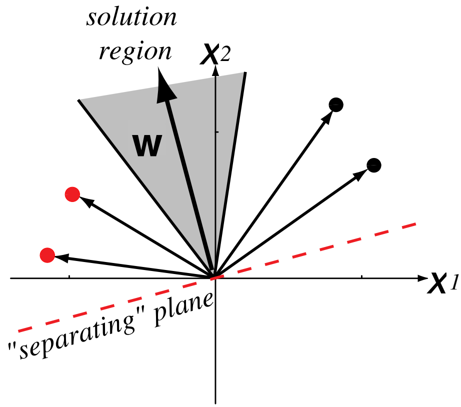
This is a general scheme, we do not know $J(\mathbf{w})$, yet.
We're looking into *error-based classification* methods: missclassified examples are used to tune the classifier...
We already discussed (stochastic) Gradient descent when talking about $Q-$function learning

Learning **w** - Perceptron criterion

Goal: Find a weight vector $\mathbf{w} \in \Re^{D+1}$ (original feature space dimensionality is D) such that:

$$\mathbf{w}^\top \mathbf{x}_j > 0 \qquad (\forall j \in \{1, 2, ..., m\})$$





(Perceptron) Criterion to be minimized:

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}} -\mathbf{w}^\top \mathbf{x}$$

where $\mathcal{X}$ is a set of missclassified $\mathbf{x}$.

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}} -\mathbf{x}$$

Notes

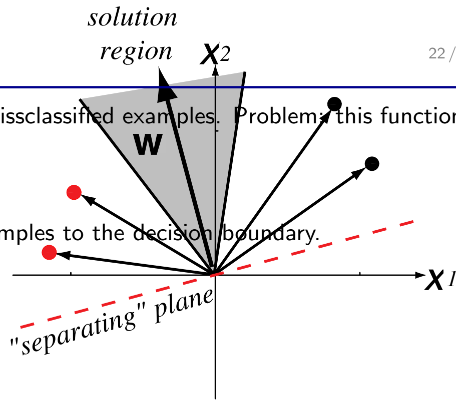
What are the possible choices for $J(\mathbf{w})$? First choice: number of missclassified examples. Problem: this function is piecewise constant.

Better choice: perceptron criterion function.

Mind that $\mathbf{w}^\top \mathbf{x}_j \leq 0$ for $\mathbf{x} \in \mathcal{X}$

Geometrically: $J(\mathbf{w}) \propto$ sum of the distance of the missclassified samples to the decision boundary.

What is $\nabla J(\mathbf{w})$ equal to?



Learning $\mathbf{w}$ - Perceptron criterion

Goal: Find a weight vector $\mathbf{w} \in \Re^{D+1}$ (original feature space dimensionality is D) such that:

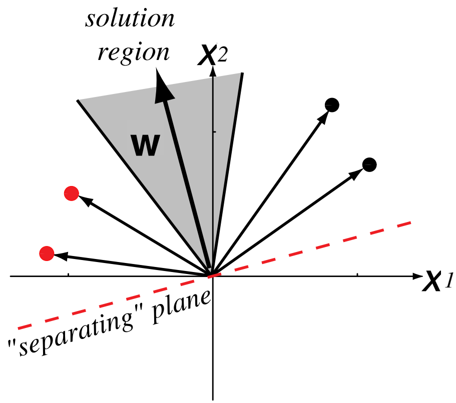
$$\mathbf{w}^\top \mathbf{x}_j > 0 \qquad (\forall j \in \{1, 2, ..., m\})$$

(Perceptron) Criterion to be minimized:

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}} -\mathbf{w}^\top \mathbf{x}$$

where $\mathcal{X}$ is a set of missclassified $\mathbf{x}$.

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}} -\mathbf{x}$$



Notes

What are the possible choices for $J(\mathbf{w})$? First choice: number of missclassified examples. Problem: this function is piecewise constant.
Better choice: perceptron criterion function.
Mind that $\mathbf{w}^\top \mathbf{x}_j \leq 0$ for $\mathbf{x} \in \mathcal{X}$
Geometrically: $J(\mathbf{w}) \propto$ sum of the distance of the missclassified samples to the decision boundary.
What is $\nabla J(\mathbf{w})$ equal to?

(Batch) Perceptron algorithm

Initialize $\mathbf{w}$, threshold θ, learning rate α
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha(k) \sum_{\mathbf{x} \in \mathcal{X}(k)} \mathbf{x}$
until $|\alpha(k) \sum_{\mathbf{x} \in \mathcal{X}(k)} \mathbf{x}| < \theta$
return $\mathbf{w}$

Notes

Next weight vector $\sim$ adding some multiple of the sum of the missclassified samples to the present weight vector.

Fixed-increment single-sample Perceptron

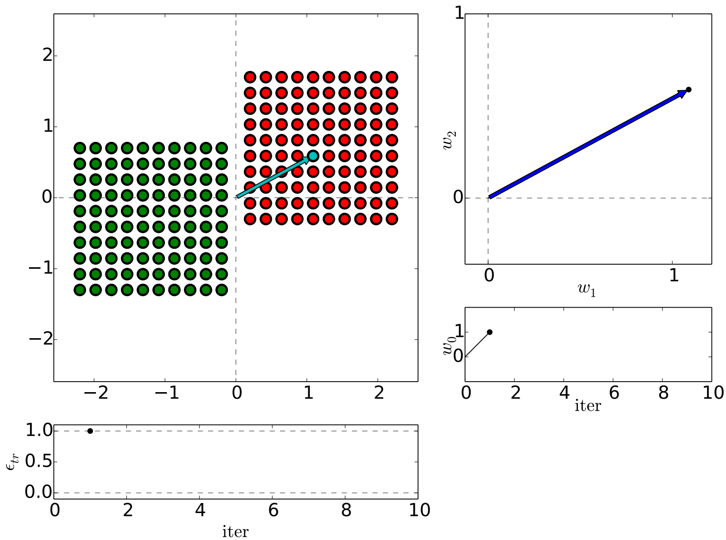
n patterns/samples, we are looping over all patterns repeatedly

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then** $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

Notes

As we are looping over all patterns repeatedly, it is not an on-line algorithm

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

Initialize **w**
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then**
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all **x** correctly classified
return **w**

(Dark) Blue is **w** after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of **x**.

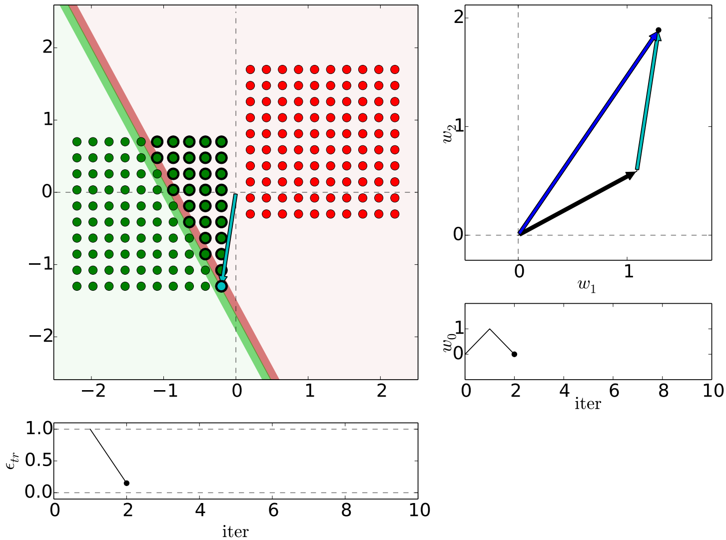
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red **x** are $+$, green are $-$

Track the iteration steps. After each update **x**, draw a separating line for the next and verify.

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

> Initialize $\mathbf{w}$
> $k \leftarrow 0$
> **repeat**
> $k \leftarrow (k+1) \bmod n$
>   **if** $\mathbf{x}^k$ missclassified, **then**
> $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
> **until** all $\mathbf{x}$ correctly classified
> return $\mathbf{w}$

(Dark) Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

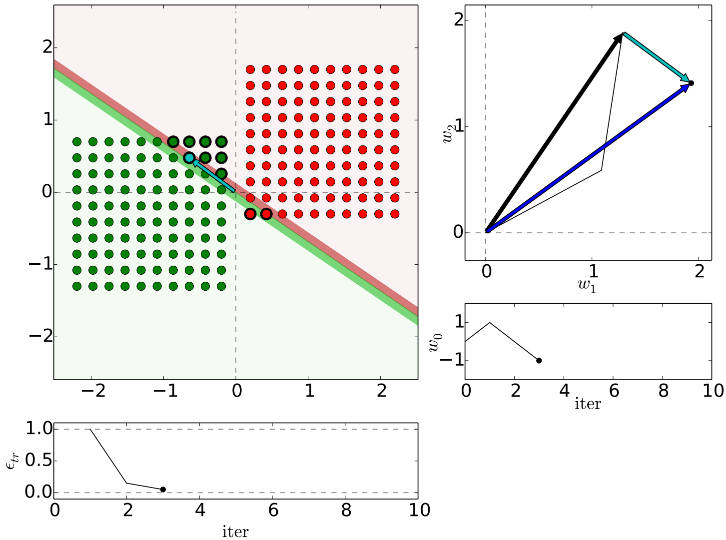
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then**
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

(Dark) Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

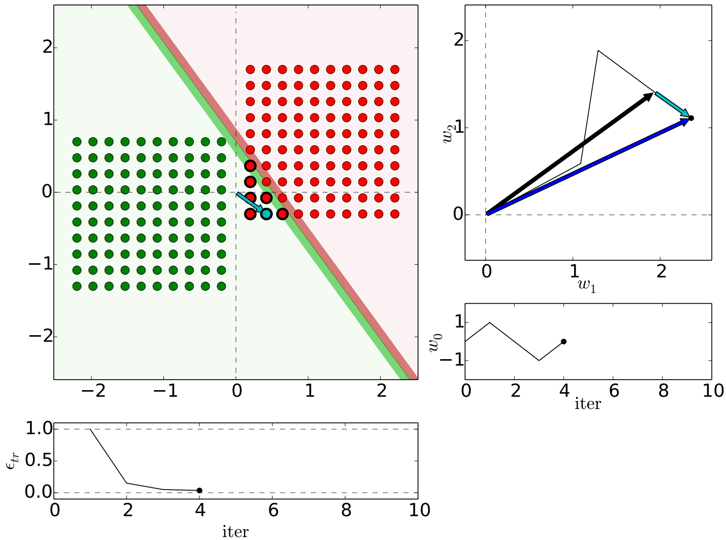
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then**
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

(Dark) Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

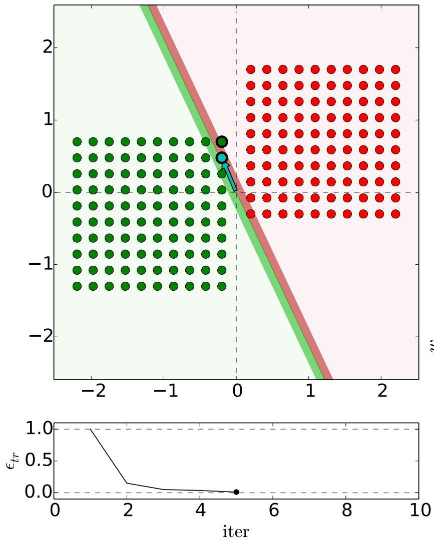
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

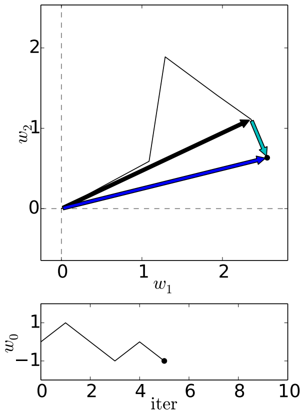
$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Perceptron iterations/loops





n patterns/samples, we are looping over all patterns repeatedly:

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then**
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

(Dark) Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

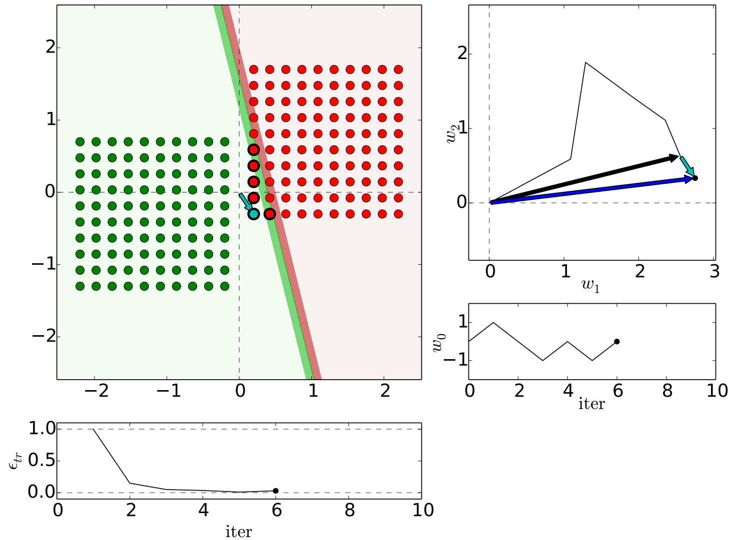
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
$\quad k \leftarrow (k+1) \bmod n$
$\quad$ **if** $\mathbf{x}^k$ missclassified, **then**
$\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

(Dark) Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

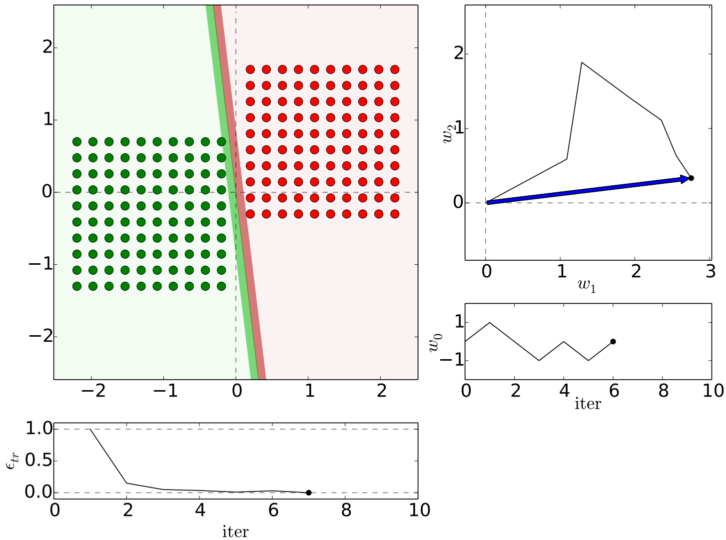
$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

(as discussed few slides ago)
Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Perceptron iterations/loops



n patterns/samples, we are looping over all patterns repeatedly:

Initialize $\mathbf{w}$
$k \leftarrow 0$
repeat
  $k \leftarrow (k+1) \bmod n$
  if $\mathbf{x}^k$ missclassified, **then**
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^k$
until all $\mathbf{x}$ correctly classified
return $\mathbf{w}$

(Dark)

Blue is $\mathbf{w}$ after update step. Reds are $+$, Greens $-$.

Notes

Keep in mind the $\pm$ normalization of $\mathbf{x}$.

$$g(\mathbf{x}) = \begin{cases} s = 1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 > 0\,, \\ s = -1\,, & \text{if} \quad \mathbf{w}^\top \mathbf{x} + w_0 < 0\,. \end{cases}$$

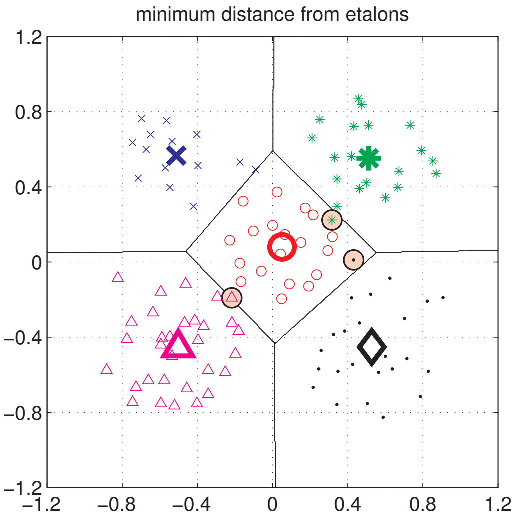
$$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}, \mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

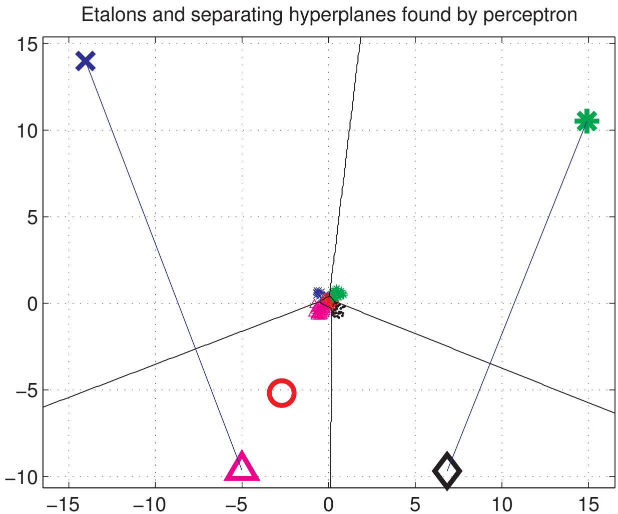
(as discussed few slides ago)

Red $\mathbf{x}$ are $+$, green are $-$

Track the iteration steps. After each update $\mathbf{x}$, draw a separating line for the next and verify.

Etalons: means vs. found by perceptron

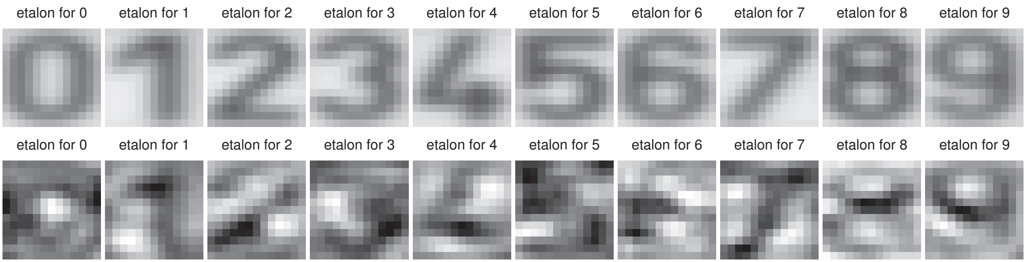




Figures from [5]

Notes

Digit recognition - etalons means vs. perceptron

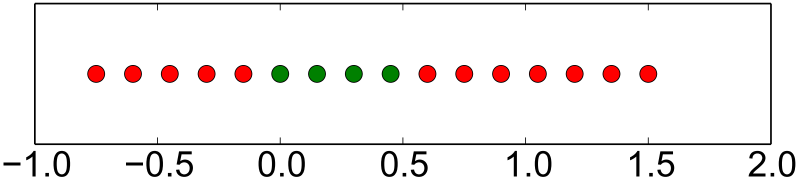


Figures from [5]

Notes

"Prototypes" resulting from the perceptron algorithm are harder to interpret because they are not means – instead, they are optimized for separating the classes.

What if not lin separable?

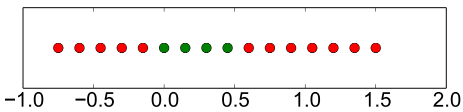


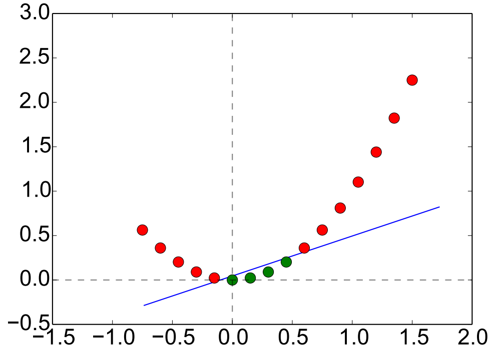
Dimension lifting

$$\mathbf{x} = [x, x^2]^\top$$

Notes

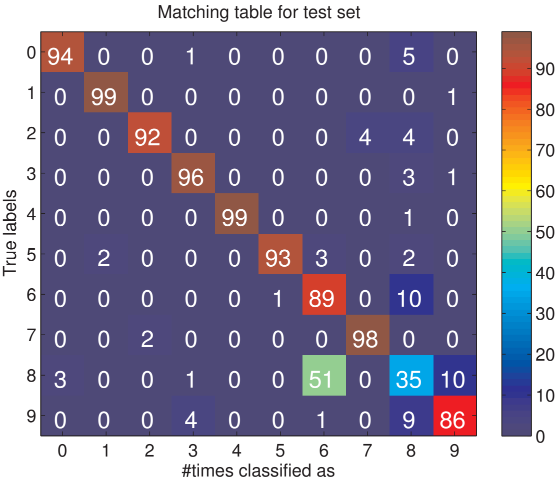
Dimension lifting, $\mathbf{x} = [x, x^2]^\top$

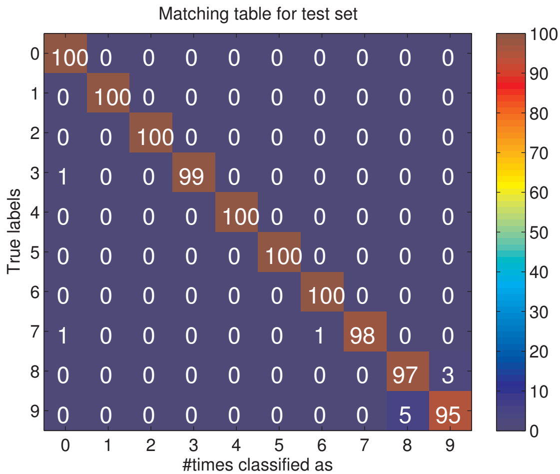




Notes

Performance comparison, parameters fixed

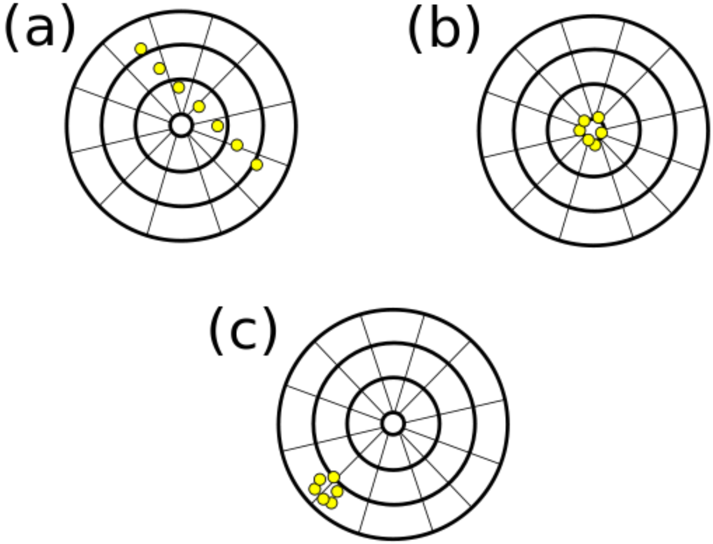




Notes

Why there some errors in perceptron results? We said zero error on training set.

Accuracy vs precision



https://commons.wikimedia.org/wiki/File:Precision_versus_accuracy.svg

Notes

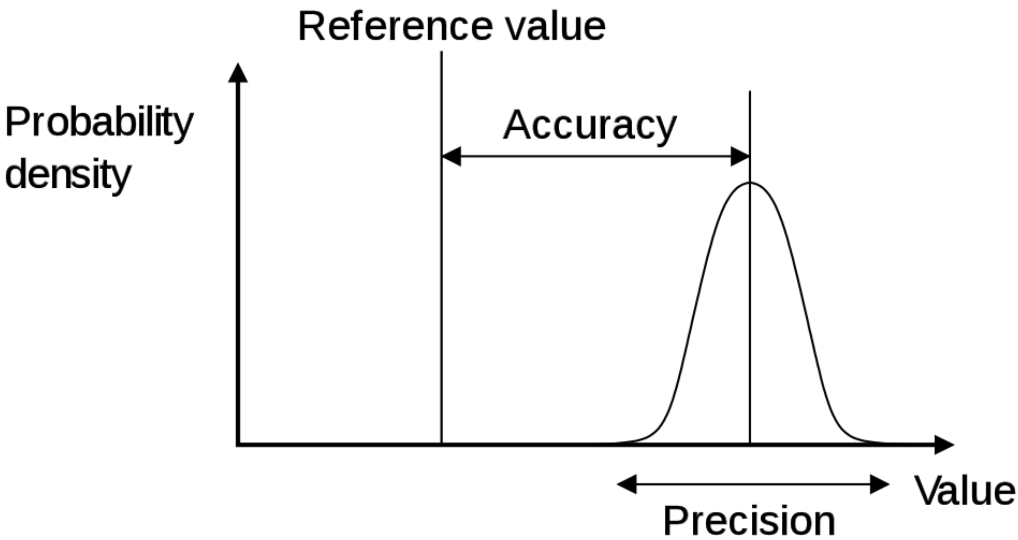
Accuracy: how close (is your model) to the truth. Precision: how consistent/stable
In German:

- Accuracy: Richtigkeit
- Precision: Präzision
- Both together: Genauigkeit

In Czech:

- Accuracy: Věrnost, přesnost.
- Precition: Rozptyl,

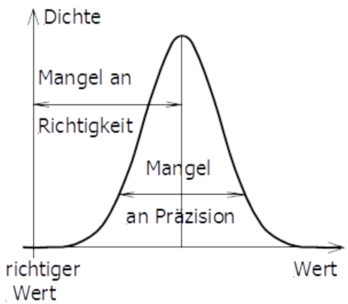
Accuracy vs precision



https://en.wikipedia.org/wiki/Accuracy_and_precision

Notes

Accuracy: how close (is your model) to the truth. Precision: how consistent/stable.
Think about terms *bias* and *error*. I



References I

Further reading: Chapter 18 of [4], or chapter 4 of [1], or chapter 5 of [2]. Many Matlab figures created with the help of [3]. You may also play with demo functions from [5].

[1] Christopher M. Bishop.
Pattern Recognition and Machine Learning.
Springer Science+Bussiness Media, New York, NY, 2006.
PDF freely downloadable.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork.
Pattern Classification.
John Wiley & Sons, 2nd edition, 2001.

[3] Votjěch Franc and Václav Hlaváč.
Statistical pattern recognition toolbox.
http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html.

Notes

References II

[4] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[5] Tomáš Svoboda, Jan Kybic, and Hlaváč Václav.
Image Processing, Analysis and Machine Vision — A MATLAB Companion.
Thomson, Toronto, Canada, 1st edition, September 2007.
http://visionbook.felk.cvut.cz/.

Notes