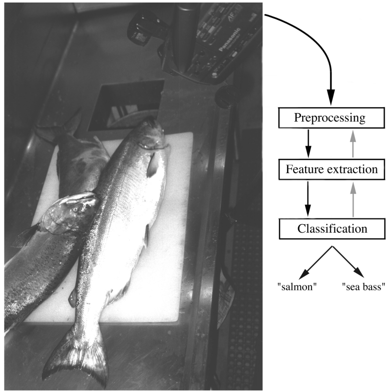
Classifiers, intro, evaluation

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

Department of Cybernetics, Vision for Robotics and Autonomous Systems,
Center for Machine Perception (CMP)

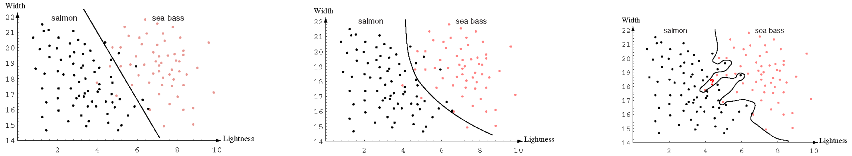
May 22, 2018

Classification example: What's the fish?



- Factory for fish processing
- 2 classes $s_{1,2}$:
  - salmon
  - sea bass
- Features $\vec{x}$: length, width, lightness etc. from a camera

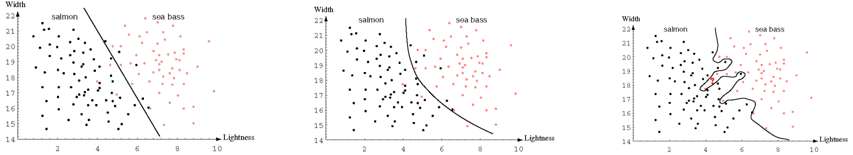
Fish classification in feature space



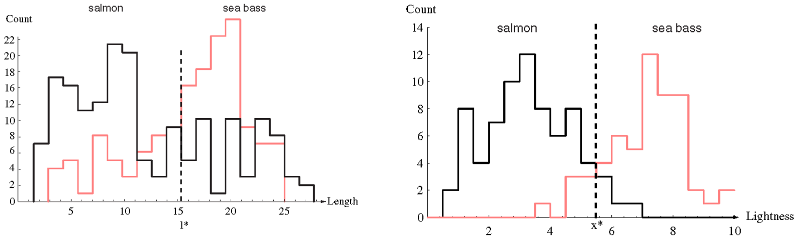
- Linear, quadratic, k-nearest neighbor classifier

- Feature frequency per class shown using histograms
- Classification errors due to histogram overlap

Fish classification in feature space



- Linear, quadratic, k-nearest neighbor classifier



- Feature frequency per class shown using histograms
- Classification errors due to histogram overlap

Fish – classification using probability

$$posterior = \frac{likelihood \times prior}{evidence}$$

- Notation for classification problem
  - Classes $s_j \in S$ (e.g., salmon, sea bass)
  - Features $x_i \in X$ or feature vectors ($\vec{x}_i$) (also called attributes)
- Optimal classification of $\vec{x}$:

$$\delta^*(\vec{x}) = \arg\max_j P(s_j|\vec{x})$$

- We thus choose the most probable class for a given feature vector.
- Both likelihood and prior are taken into account – recall Bayes rule:

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

Fish – classification using probability

$$posterior = \frac{likelihood \times prior}{evidence}$$

- Notation for classification problem
  - Classes $s_j \in S$ (e.g., salmon, sea bass)
  - Features $x_i \in X$ or feature vectors $(\vec{x}_i)$ (also called attributes)
- Optimal classification of $\vec{x}$:

$$\delta^*(\vec{x}) = \arg\max_j P(s_j|\vec{x})$$

- We thus choose the most probable class for a given feature vector.
- Both likelihood and prior are taken into account – recall Bayes rule:

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

Bayes classification in practice

- Usually we are not given $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, training examples $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \ldots (\vec{x}_l, s_l)$
  - so-called i.i.d (independent, identically distributed) multiset
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$
- Without knowing anything about the distribution, a non-parametric estimate:

$$P(s|\vec{x}) \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \text{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

- Hard in practice:
  - To reliably estimate $P(s|\vec{x})$, the number of examples grows exponentially with the number of elements of $\vec{x}$.
    - e.g. with the number of pixels in images
    - curse of dimensionality
    - denominator often 0

Why hard? Way too many various $\vec{x}$. Think about simple binary 10×10 image - $\vec{x}$ contains $0, 1$, position matters. What is the total number of unique images? Think binary, 1×8 binary image?

Bayes classification in practice

- Usually we are not given $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, training examples $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \dots (\vec{x}_l, s_l)$
  - so-called i.i.d (independent, identically distributed) multiset
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$
- Without knowing anything about the distribution, a non-parametric estimate:

$$P(s|\vec{x}) \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \text{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

- Hard in practice:
  - To reliably estimate $P(s|\vec{x})$, the number of examples grows exponentially with the number of elements of $\vec{x}$.
    - e.g. with the number of pixels in images
    - curse of dimensionality
    - denominator often 0

Why hard? Way too many various $\vec{x}$. Think about simple binary 10×10 image - $\vec{x}$ contains $0, 1$, position matters. What is the total number of unique images? Think binary, 1×8 binary image?

Bayes classification in practice

- Usually we are not given $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, training examples $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \ldots (\vec{x}_l, s_l)$
  - so-called i.i.d (independent, identically distributed) multiset
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$
- Without knowing anything about the distribution, a non-parametric estimate:

$$P(s|\vec{x}) \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \text{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

- Hard in practice:
  - To reliably estimate $P(s|\vec{x})$, the number of examples grows exponentially with the number of elements of $\vec{x}$.
    - e.g. with the number of pixels in images
    - curse of dimensionality
    - denominator often 0

Why hard? Way too many various $\vec{x}$. Think about simple binary 10×10 image - $\vec{x}$ contains $0, 1$, position matters. What is the total number of unique images? Think binary, 1×8 binary image?

Bayes classification in practice

- Usually we are not given $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, training examples $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \dots (\vec{x}_l, s_l)$
  - so-called i.i.d (independent, identically distributed) multiset
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$
- Without knowing anything about the distribution, a non-parametric estimate:

$$P(s|\vec{x}) \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \text{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

- Hard in practice:
  - To reliably estimate $P(s|\vec{x})$, the number of examples grows exponentially with the number of elements of $\vec{x}$.
    - e.g. with the number of pixels in images
    - curse of dimensionality
    - denominator often 0

Why hard? Way too many various $\vec{x}$. Think about simple binary 10×10 image - $\vec{x}$ contains $0, 1$, position matters. What is the total number of unique images? Think binary, 1×8 binary image?

Naïve Bayes classification

- For efficient classification we must thus rely on additional assumptions.
- In the exceptional case of statistical independence between $\vec{x}$ components for each class s it holds

$$P(\vec{x}|s) = P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots$$

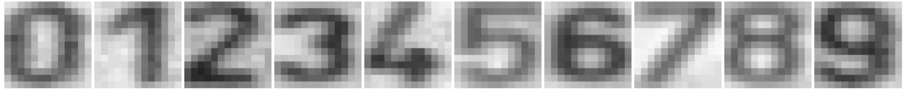
- Use simple Bayes law and maximize:

$$P(s|\vec{x}) = \frac{P(\vec{x}|s)P(s)}{P(\vec{x})} = \frac{P(s)}{P(\vec{x})} P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots =$$

- No combinatorial curse in estimating $P(s)$ and $P(x[i]|s)$ separately for each i and s.
- No need to estimate $P(\vec{x})$. (Why?)
- $P(s)$ may be provided apriori.
- naïve = when used despite statistical dependence

Why naïve at all? Consider $N-$ dimensional space, $8-bit$ values. Instead of problem 8^N we have $8 \times N$ problem.
Think about statistical independence. Example1: person's weight and height. Are they independent? Example2: pixel values in images.

Example: Digit recognition



- Input: 8-bit image 13×13, intensities $0 - 255$.
- Output: Digit $0 - 9$. Decision about the class, classification.
- Features: Pixel intensities ...

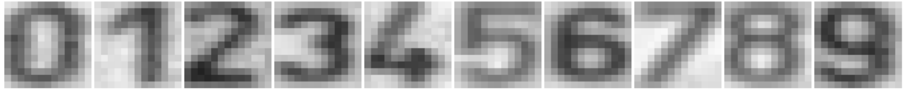
Collect data , ...

- $P(\vec{x})$. What is the dimension of $\vec{x}$? How many possible images?
- Learn $P(\vec{x}|s)$ per each class (digit).
- Classify $s^* = \operatorname{argmax}_s P(s|\vec{x})$.

We can create many more features than just pixel intensities. But first things first.

We are assuming all errors are equally important - minimizing the number of wrong decisions

Example: Digit recognition



- Input: 8-bit image 13×13, intensities $0 - 255$.
- Output: Digit $0 - 9$. Decision about the class, classification.
- Features: Pixel intensities ...

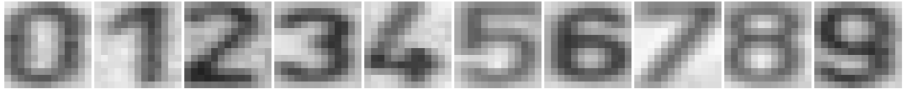
Collect data , ...

- $P(\vec{x})$. What is the dimension of $\vec{x}$? How many possible images?
- Learn $P(\vec{x}|s)$ per each class (digit).
- Classify $s^* = \text{argmax}_s P(s|\vec{x})$.

We can create many more features than just pixel intensities. But first things first.

We are assuming all errors are equally important - minimizing the number of wrong decisions

Example: Digit recognition



- Input: 8-bit image 13×13, intensities $0-255$.
- Output: Digit $0-9$. Decision about the class, classification.
- Features: Pixel intensities ...

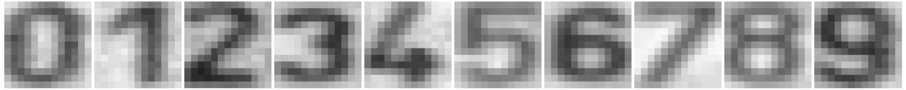
Collect data , ...

- $P(\vec{x})$. What is the dimension of $\vec{x}$? How many possible images?
- Learn $P(\vec{x}|s)$ per each class (digit).
- Classify $s^* = \mathrm{argmax}_s P(s|\vec{x})$.

We can create many more features than just pixel intensities. But first things first.

We are assuming all errors are equally important - minimizing the number of wrong decisions

Example: Digit recognition



- Input: 8-bit image 13×13, intensities $0 - 255$.
- Output: Digit $0 - 9$. Decision about the class, classification.
- Features: Pixel intensities ...

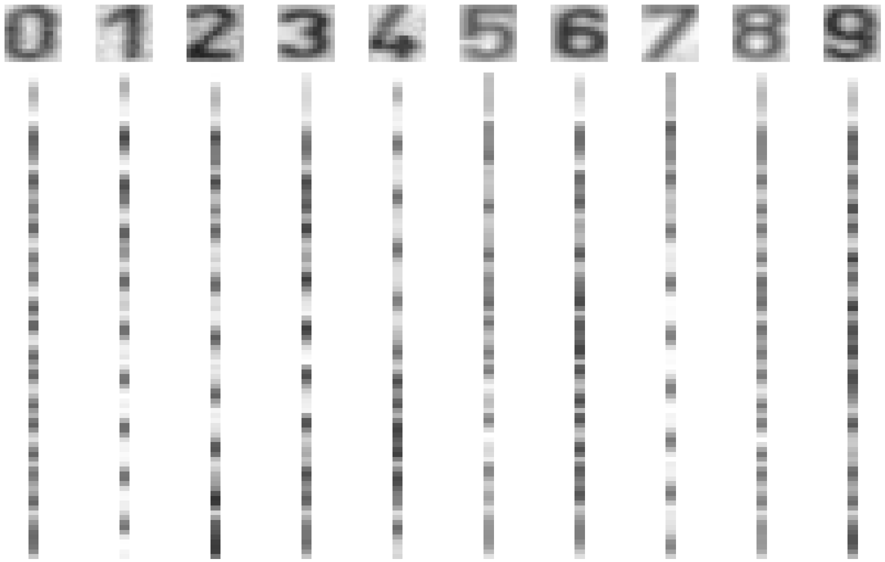
Collect data , ...

- $P(\vec{x})$. What is the dimension of $\vec{x}$? How many possible images?
- Learn $P(\vec{x}|s)$ per each class (digit).
- Classify $s^* = \text{argmax}_s P(s|\vec{x})$.

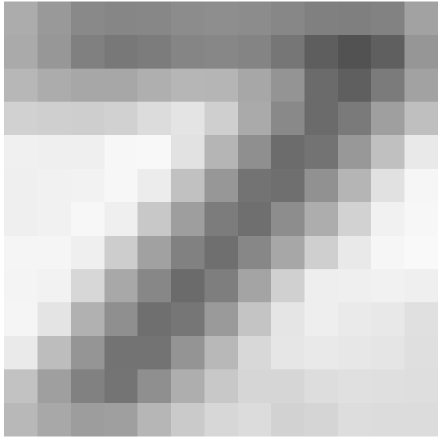
We can create many more features than just pixel intensities. But first things first.

We are assuming all errors are equally important - minimizing the number of wrong decisions

From images to $\vec{x}$

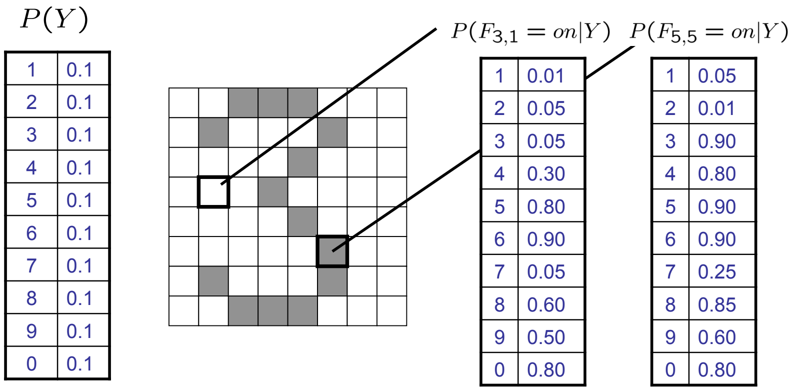


Conditional probabilities



- Apriori digit probabilities $P(s_k)$
- Likelihoods for pixels. $P(x_{u,v} = I_i | s_k)$

We can rearrange pixels into vector - then using a linear index $P(x_j = I_i | s_k)$.



Y	$P(Y)$	$P(F_{3,1} = on\|Y)$	$P(F_{5,5} = on\|Y)$
1	0.1	0.01	0.05
2	0.1	0.05	0.01
3	0.1	0.05	0.90
4	0.1	0.30	0.80
5	0.1	0.80	0.90
6	0.1	0.90	0.90
7	0.1	0.05	0.25
8	0.1	0.60	0.85
9	0.1	0.50	0.60
0	0.1	0.80	0.80

image by courtesy of P. Abeel, http://ai.berkeley.edu

Training and testing

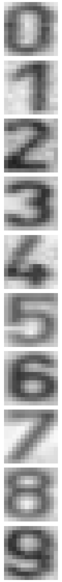
Data labeled instances.

- Training set
- Held-out (validation) set
- Testing set.

Features : Attribute-value pairs.

Learning cycle:

- Learn parameters (e.g. probabilities) on training set.
- Tune hyperparameters on held-out (validation) set.
- Evaluate performance on testing set.



We will be talking about hyperparameters in a minute

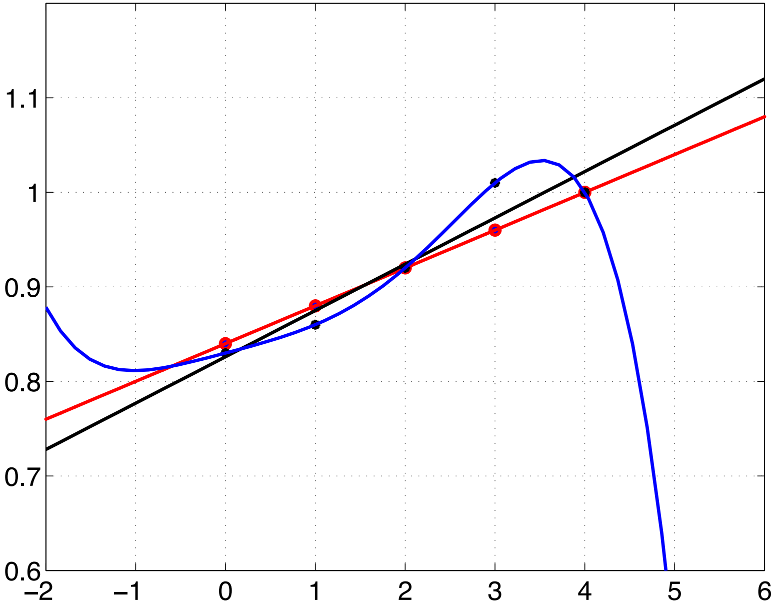
Generalization and overfiting

- Data: training, validation, testing. Wanted classifier performs well on what data?
- Overfitting: too close to training, poor on testing

Generalization and overfiting

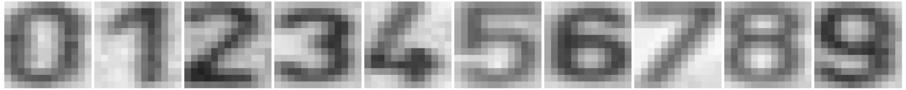
- Data: training, validation, testing. Wanted classifier performs well on what data?
- Overfitting: too close to training, poor on testing

Overfiting

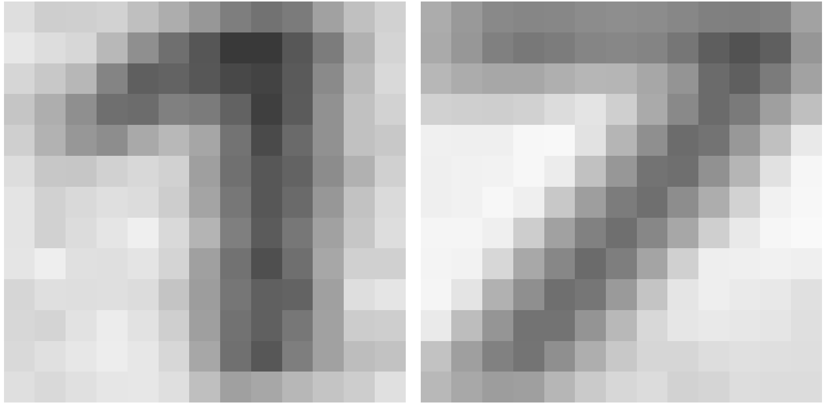


see the `overfit.m` demo

Unseen events



Images 13×13, intensities $0-255$, 100 exemplars per each class.



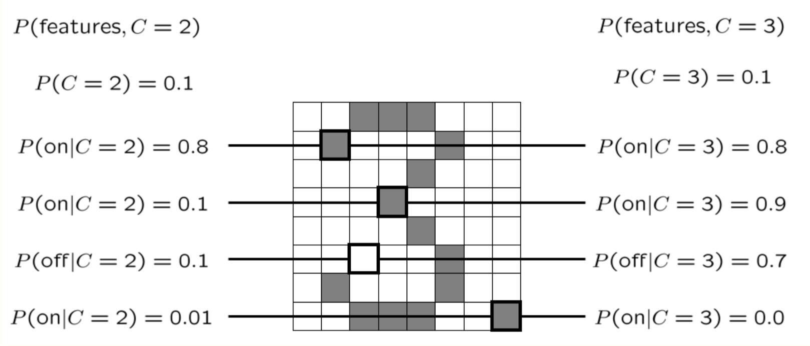


image by courtesy of P. Abeel, http://ai.berkeley.edu

Laplace smoothing

$$P(x) = \frac{\text{count}(x)}{\text{total samples}}$$

Problem: $\text{count}(x) = 0$

Pretend you see the sample one more time.

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{\sum_x [c(x) + 1]}$$

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{N + |X|}$$

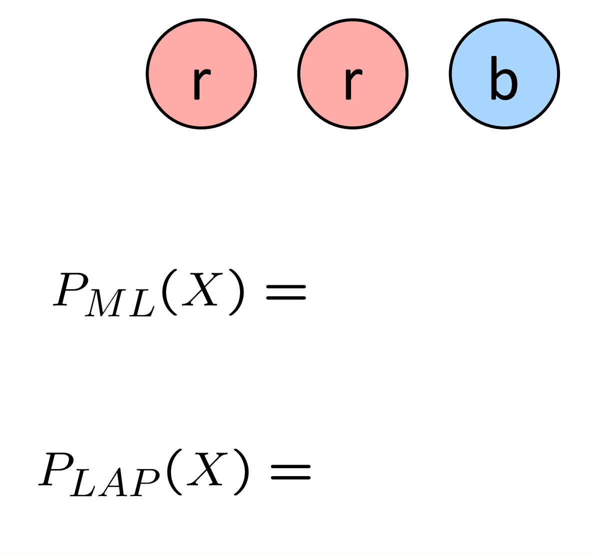


image by courtesy of P. Abeel, http://ai.berkeley.edu

Laplace smoothing

$$P(x) = \frac{\text{count}(x)}{\text{total samples}}$$

Problem: $\text{count}(x) = 0$
Pretend you see the sample one more time.

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{\sum_x [c(x) + 1]}$$

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{N + |X|}$$

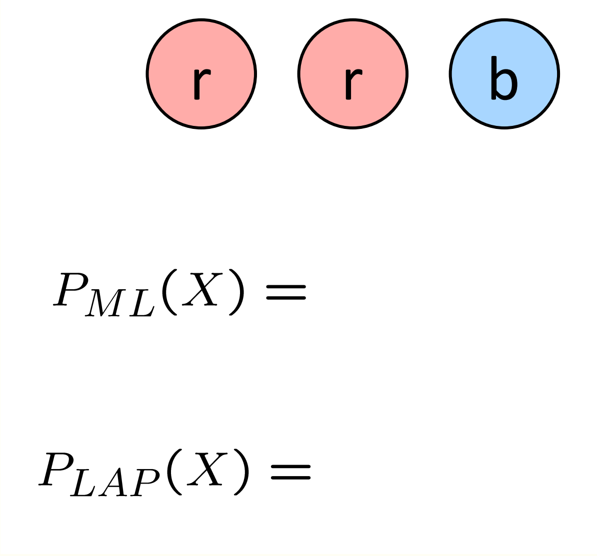


image by courtesy of P. Abeel, http://ai.berkeley.edu

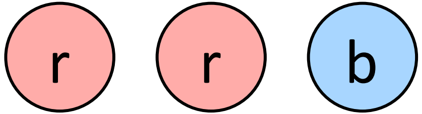
Laplace smoothing

$$P(x) = \frac{\text{count}(x)}{\text{total samples}}$$

Problem: $\text{count}(x) = 0$
Pretend you see the sample one more time.

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{\sum_x [c(x) + 1]}$$

$$P_{\text{LAP}}(x) = \frac{c(x) + 1}{N + |X|}$$



$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

image by courtesy of P. Abeel, http://ai.berkeley.edu

Laplace smoothing - as a hyperparameter k

Pretend you see every sample k extra times:

$$P_{\text{LAP}}(x) = \frac{c(x) + k}{\sum_x [c(x) + k]}$$

$$P_{\text{LAP}}(x) = \frac{c(x) + k}{N + k|X|}$$

For conditional, smooth each condition independently

$$P_{\text{LAP}}(x|s) = \frac{c(x, s) + k}{c(s) + k|X|}$$

Product of many small numbers ...

$$P(s|\vec{x}) = \frac{P(\vec{x}|s)P(s)}{P(\vec{x})} = \frac{P(s)}{P(\vec{x})} P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots$$

$P(\vec{x})$ not needed, ...

$$\log(P(x[1]|s)P(x[2]|s)\cdots) = log(P(x[1]|s)) + log(P(x[2]|s)) + \cdots$$

just try `prod(rand(1,100))` and `prod(rand(1,10000))` in Matlab.
What is the way out?

Product of many small numbers ...

$$P(s|\vec{x}) = \frac{P(\vec{x}|s)P(s)}{P(\vec{x})} = \frac{P(s)}{P(\vec{x})} P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots$$

$P(\vec{x})$ not needed, ...

$$\log(P(x[1]|s)P(x[2]|s)\cdots) = log(P(x[1]|s)) + log(P(x[2]|s)) + \cdots$$

just try `prod(rand(1,100))` and `prod(rand(1,10000))` in Matlab.
What is the way out?

Inference and decision

Inference stage - learning models/function/parameters from data.
Decision stage - decide about a query $\vec{x}$.

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

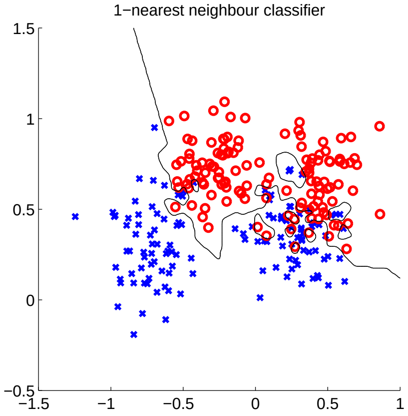
Generative models because by sampling from them it is possible to generate synthetic data points $\vec{x}$.
For the discriminative model one can consider, e.g. logistic function:

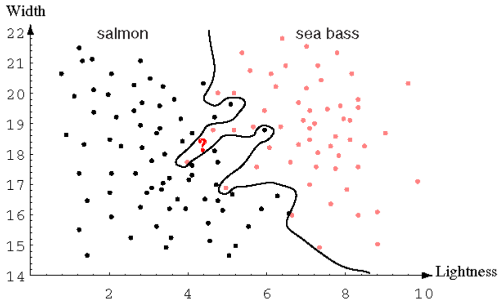
$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$

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.





$K-$ Nearest Neighbor and Bayes

Assume data:

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

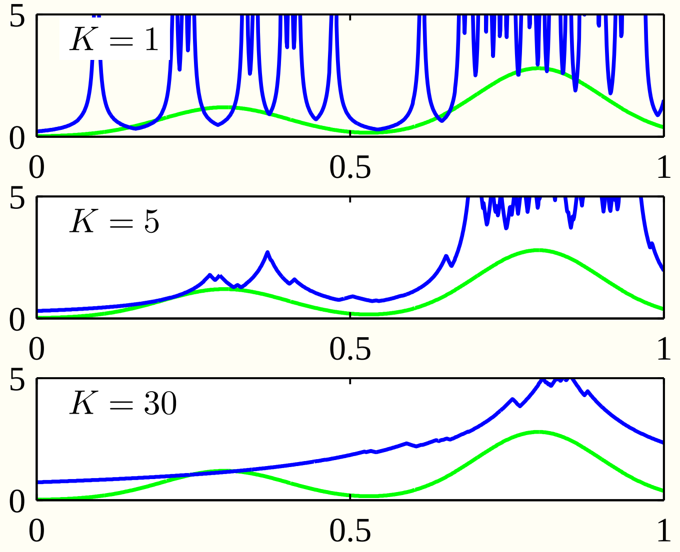
We want classify $\vec{x}$. We draw a sphere centered at $\vec{x}$ containing K points irrespective of class. V is the volume of this sphere.

$$P(\vec{x}) = \frac{K}{NV}$$

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

$$P(s_j) = \frac{N_j}{N}$$

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



$K-$ Nearest Neighbor and Bayes

Assume data:

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

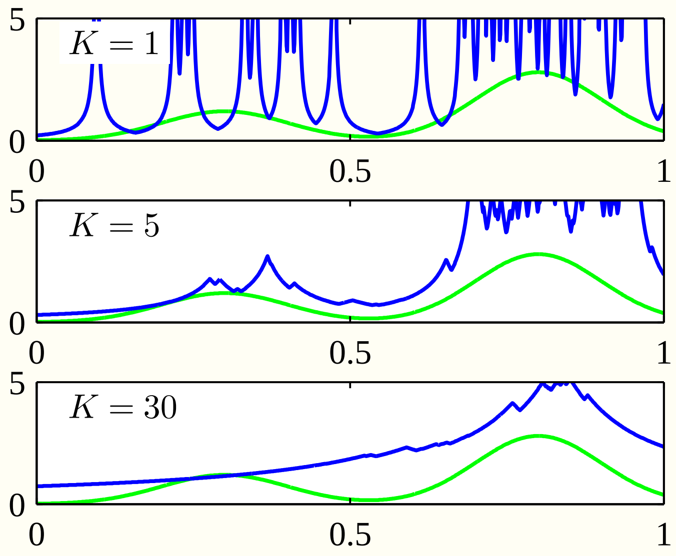
We want classify $\vec{x}$. We draw a sphere centered at $\vec{x}$ containing K points irrespective of class. V is the volume of this sphere.

$$P(\vec{x}) = \frac{K}{NV}$$

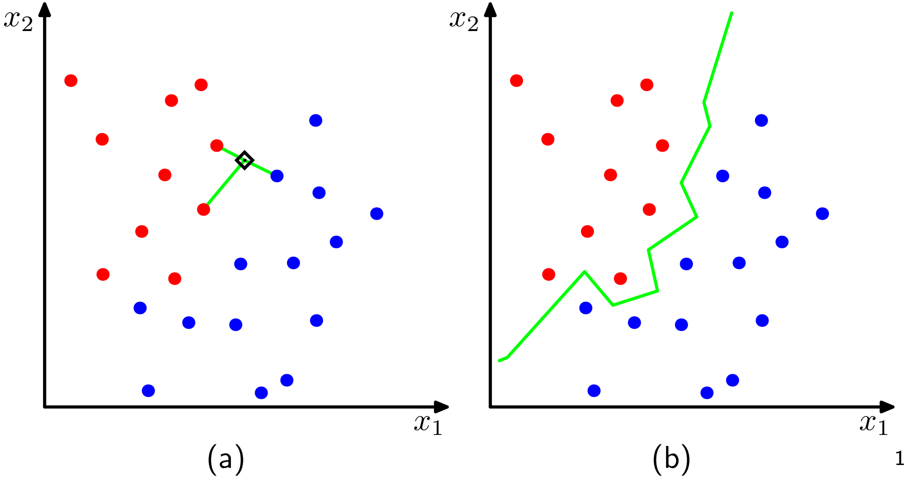
$$P(\vec{x}|s_j) = \frac{K_j}{N_j V}$$

$$P(s_j) = \frac{N_j}{N}$$

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



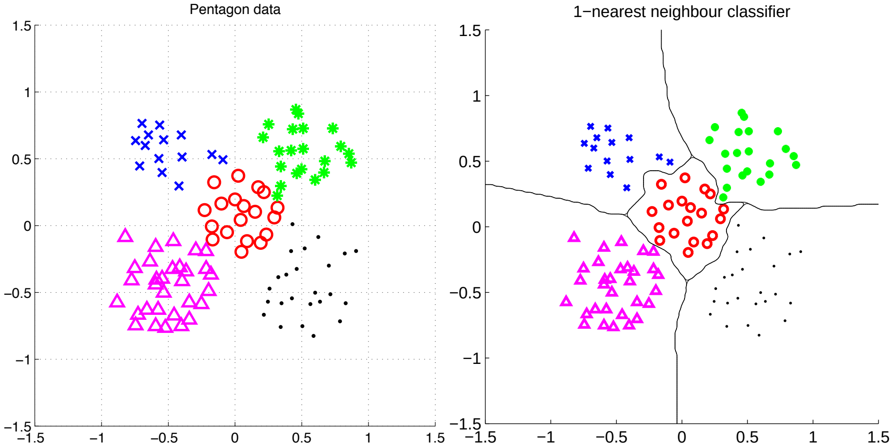
NN classification example



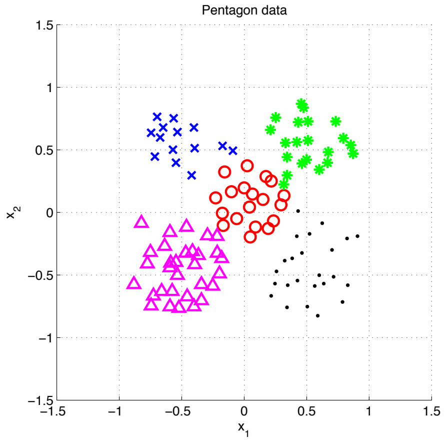
1

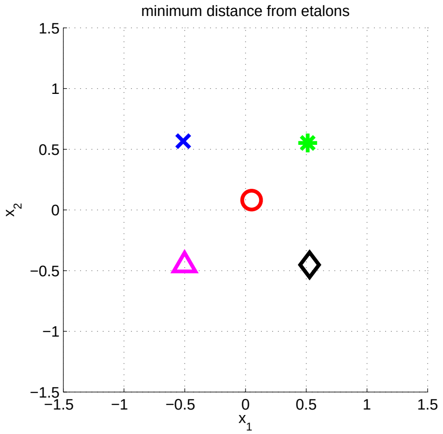
1 Figs from [1]

NN classification example



Etalon based classification

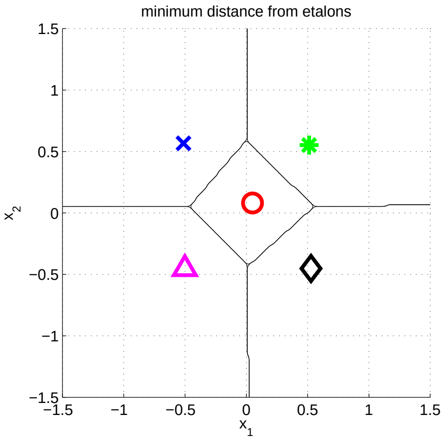




Represent $\vec{x}$ by etalon , $\vec{e}_s$ per each class $s \in S$

Separate etalons

$$f(\vec{x}) = \arg\min_{s \in S}(||\vec{x} - \vec{e}_s||^2 + o_s)$$

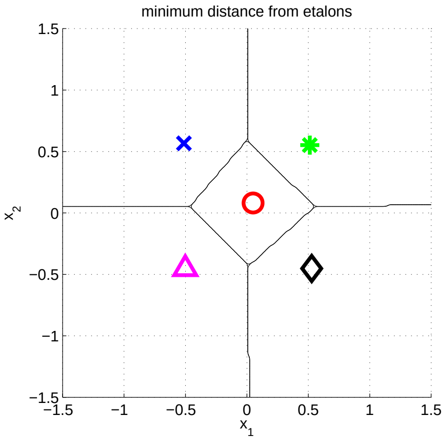


What etalons?

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

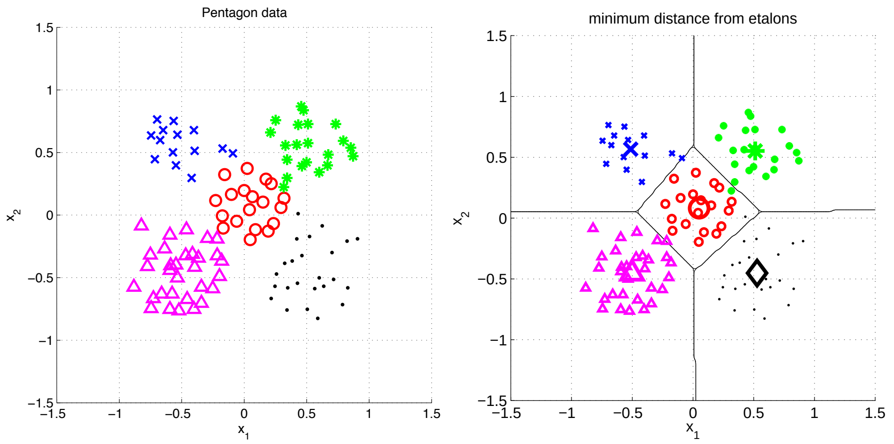
$$\vec{e}_s \stackrel{\text{def}}{=} \vec{\mu}_s = \frac{1}{|\mathcal{X}^s|}\sum_{i\in\mathcal{X}^s}\vec{x}_i^s$$

and separating hyperplanes halve distances between pairs.



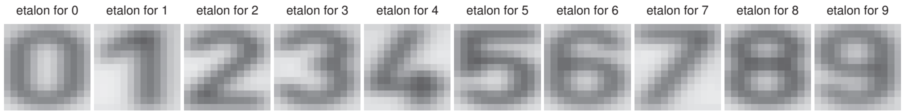
$$\mathcal{N}(\vec{x}|\vec{\mu},\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\{-\frac{1}{2}(\vec{x}-\vec{\mu})^\top\Sigma^{-1}(\vec{x}-\vec{\mu})\}$$

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



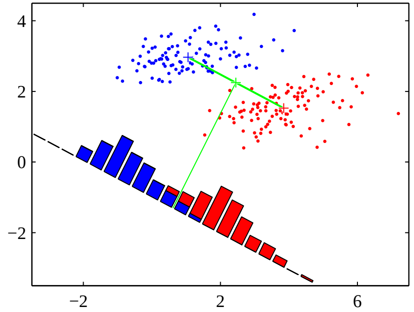
Some wrongly classified samples. We like the simple idea. Are there better etalons? How to find them?

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



Figures from [5]

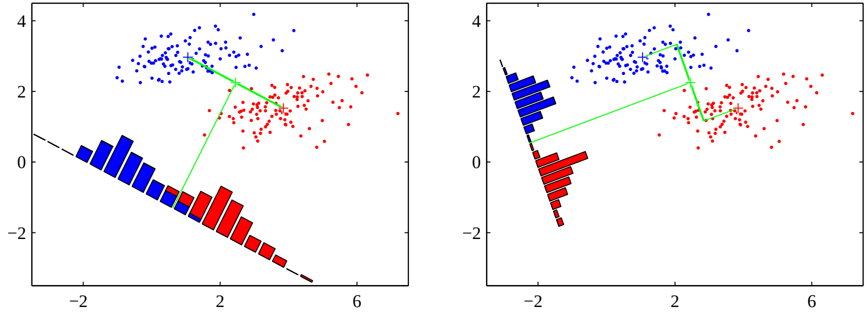
Better etalons – Fischer linear discriminant



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

Figures from [1]

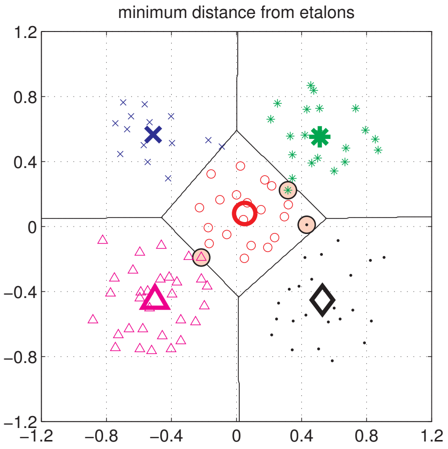
Better etalons – Fischer linear discriminant



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

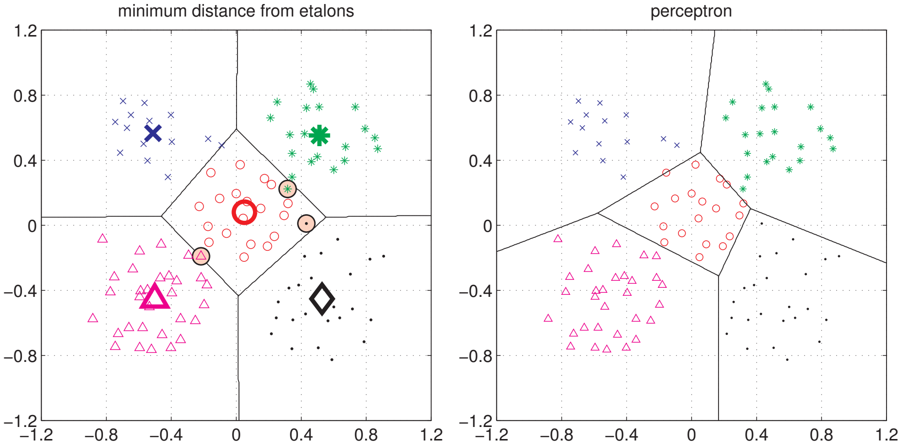
Figures from [1]

Better etalons - Perceptron



Figures from [5]

Better etalons - Perceptron



Figures from [5]

Etalon classifier – Linear classifier

$$f(\vec{x}) = \arg\min_{s \in S} \left(\|\vec{x} - \vec{e}_s\|^2 + o_s \right) = \arg\min_{s \in S} (\vec{x}^\top \vec{x} - 2\,\vec{e}_s^\top \vec{x} + \vec{e}_s^\top \vec{e}_s + o_s) =$$

$$= \arg\min_{s \in S} \left(\vec{x}^\top \vec{x} - 2\left(\vec{e}_s^\top \vec{x} - \frac{1}{2}(\vec{e}_s^\top \vec{e}_s + o_s)\right) \right) =$$

$$= \arg\min_{s \in S} (\vec{x}^\top \vec{x} - 2\,(\vec{e}_s^\top \vec{x} + b_s)) =$$

$$= \boxed{\arg\max_{s \in S} (\vec{e}_s^\top \vec{x} + b_s)} = \arg\max_{s \in S} f_s(\vec{x}). \qquad b_s = -\frac{1}{2}(\vec{e}_s^\top \vec{e}_s + o_s)$$

Linear function (plus offset)

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

Etalon classifier – Linear classifier

$$f(\vec{x}) = \arg\min_{s\in S}\left(\|\vec{x} - \vec{e}_s\|^2 + o_s\right) = \arg\min_{s\in S}(\vec{x}^\top\vec{x} - 2\,\vec{e}_s^\top\vec{x} + \vec{e}_s^\top\vec{e}_s + o_s) =$$

$$= \arg\min_{s\in S}\left(\vec{x}^\top\vec{x} - 2\left(\vec{e}_s^\top\vec{x} - \frac{1}{2}(\vec{e}_s^\top\vec{e}_s + o_s)\right)\right) =$$

$$= \arg\min_{s\in S}(\vec{x}^\top\vec{x} - 2\,(\vec{e}_s^\top\vec{x} + b_s)) =$$

$$= \boxed{\arg\max_{s\in S}(\vec{e}_s^\top\vec{x} + b_s)} = \arg\max_{s\in S} f_s(\vec{x}). \qquad b_s = -\frac{1}{2}(\vec{e}_s^\top\vec{e}_s + o_s)$$

Linear function (plus offset)

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

Etalon classifier – Linear classifier

$$f(\vec{x}) = \arg\min_{s \in S} \left(\|\vec{x} - \vec{e}_s\|^2 + o_s \right) = \arg\min_{s \in S} (\vec{x}^\top \vec{x} - 2\, \vec{e}_s^\top \vec{x} + \vec{e}_s^\top \vec{e}_s + o_s) =$$

$$= \arg\min_{s \in S} \left(\vec{x}^\top \vec{x} - 2 \left(\vec{e}_s^\top \vec{x} - \frac{1}{2} (\vec{e}_s^\top \vec{e}_s + o_s) \right) \right) =$$

$$= \arg\min_{s \in S} (\vec{x}^\top \vec{x} - 2 \, (\vec{e}_s^\top \vec{x} + b_s)) =$$

$$= \boxed{\arg\max_{s \in S} (\vec{e}_s^\top \vec{x} + b_s)} = \arg\max_{s \in S} f_s(\vec{x}). \qquad b_s = -\frac{1}{2}(\vec{e}_s^\top \vec{e}_s + o_s)$$

Linear function (plus offset)

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

Etalon classifier – Linear classifier

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

Linear function (plus offset)

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

Etalon classifier – Linear classifier

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

$$b_s = -\frac{1}{2}(\vec{e}_s^\top\vec{e}_s + o_s)$$

Linear function (plus offset)

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

Etalon classifier – Linear classifier

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

Linear function (plus offset)

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

Perceptron learning - problem set up

We seek $\mathcal{K} = \left\{(\mathbf{w}_s, w_{0s}) \mid s \in S\right\}$

$$f(\mathbf{x}) = \arg\max_{s \in S} \left(\mathbf{w}_s^\top \mathbf{x} + w_{0s}\right)$$

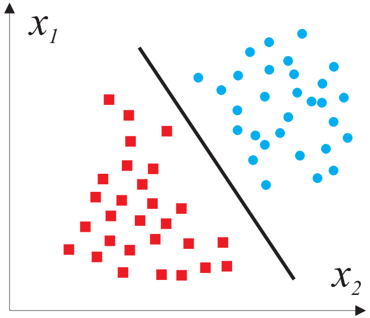
achieves no error on training set $\mathcal{T} = \left\{(\mathbf{x}^i,\, s^i),\ i = 0, 1, \ldots, m\right\}$

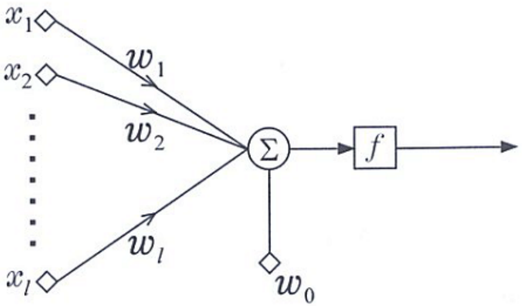
$$\epsilon_{tr} = \frac{1}{m} \sum_{j=1}^{m} \mathbf{1}\left(s^j \neq f(\mathbf{x}^j)\right), \qquad \mathbf{1}(s) = \begin{cases} 1 & s \text{ True} \\ 0 & s \text{ False} \end{cases}$$

Perceptron, two classes linearly separable

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

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





Linear seaparability - hyperplane separates/divides space into two half-spaces

Perceptron learning – Algorithm

$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}$, $\mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$ drop the dashes to avoid notation clutter.

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 algorithm (Rosenblat 1962):

1. $t \leftarrow 0$, $\mathbf{w}^{(t)} \leftarrow 0$.
2. Find a wrongly classified observation $\mathbf{x}_j$:

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

3. If there is no misclassified observation then terminate. Otherwise,

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{x}_j \,.$$

4. Goto 2.

Perceptron learning – Algorithm

$\mathbf{x}'_j = s_j \begin{bmatrix} 1 \\ \mathbf{x}_j \end{bmatrix}$, $\mathbf{w}' = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$ drop the dashes to avoid notation clutter.

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 algorithm (Rosenblat 1962):

1. $t \leftarrow 0$, $\mathbf{w}^{(t)} \leftarrow 0$.
2. Find a wrongly classified observation $\mathbf{x}_j$:

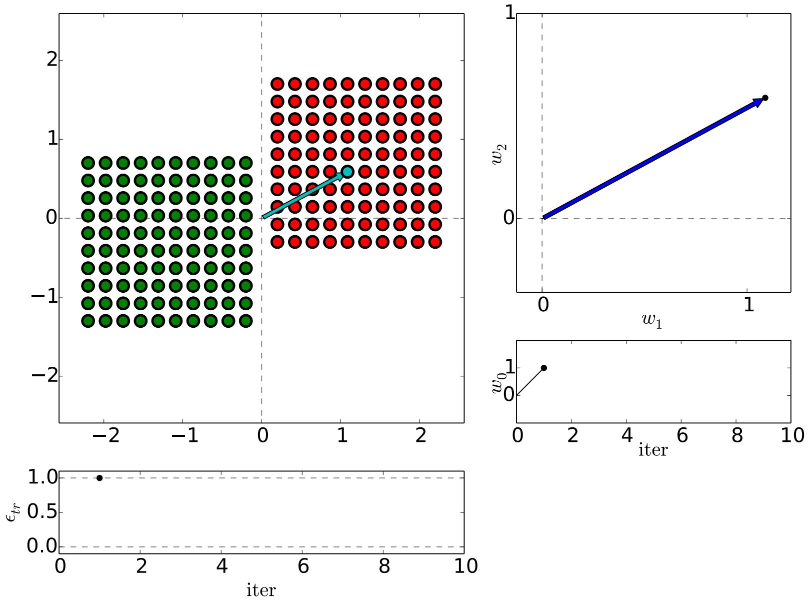
$$\mathbf{w}^{(t)\top} \mathbf{x}_j \leq 0, \qquad (j \in \{1, 2, ..., m\}.)$$

3. If there is no misclassified observation then terminate. Otherwise,

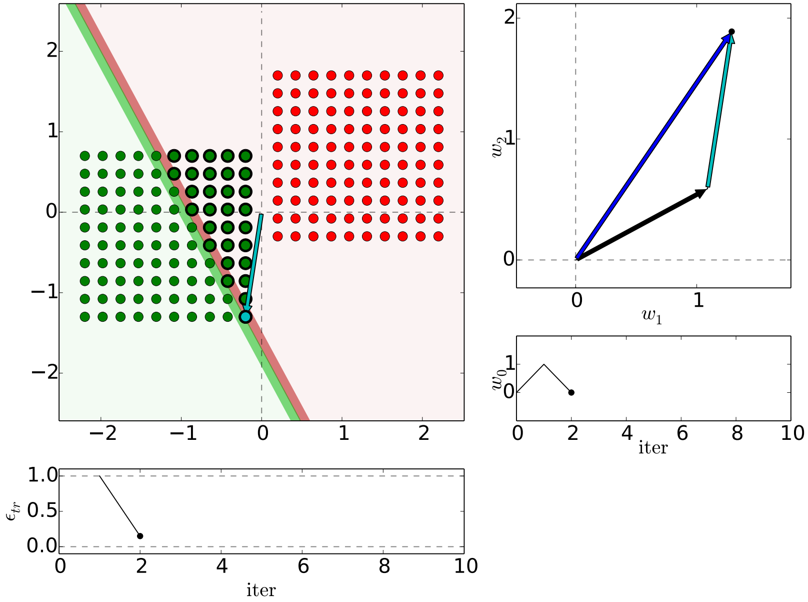
$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{x}_j \ .$$

4. Goto 2.

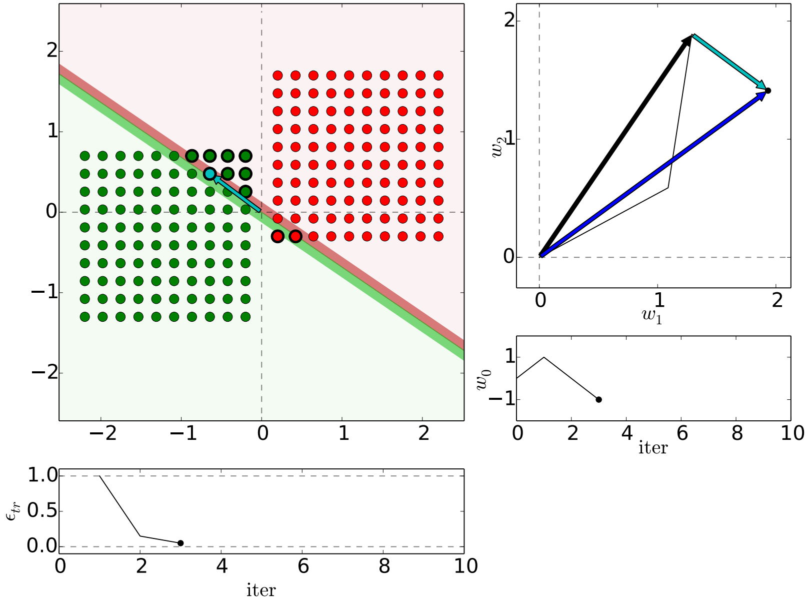
Perceptron iterations



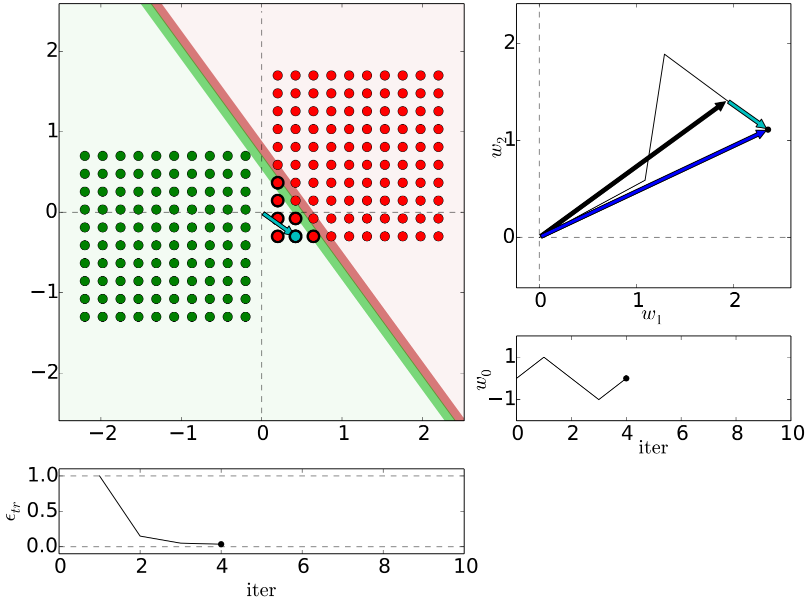
Perceptron iterations



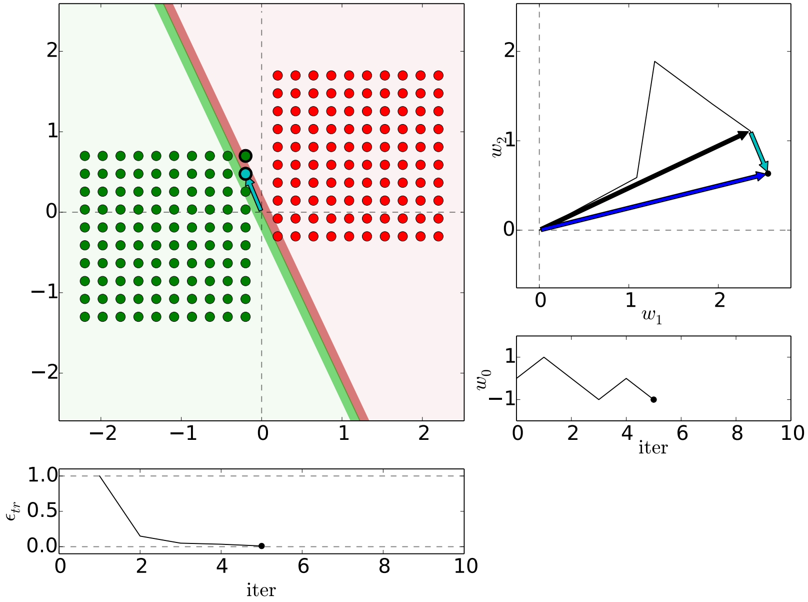
Perceptron iterations



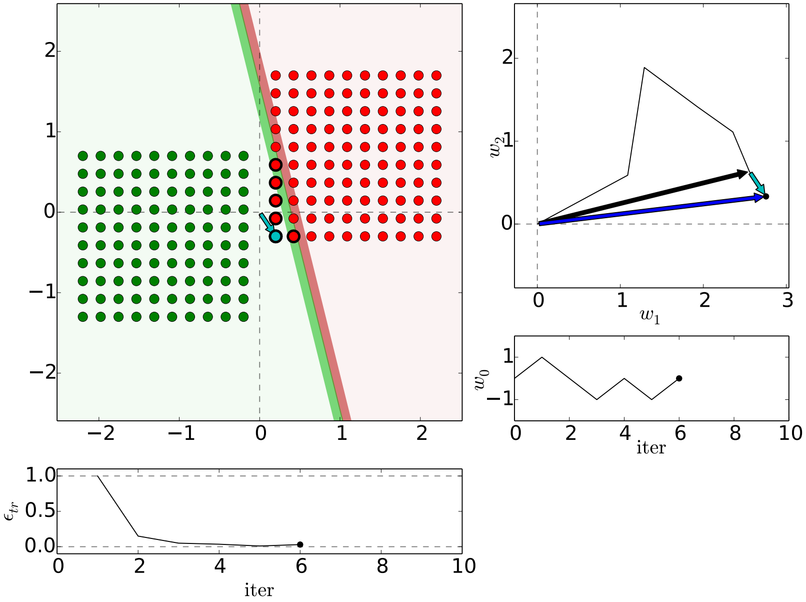
Perceptron iterations



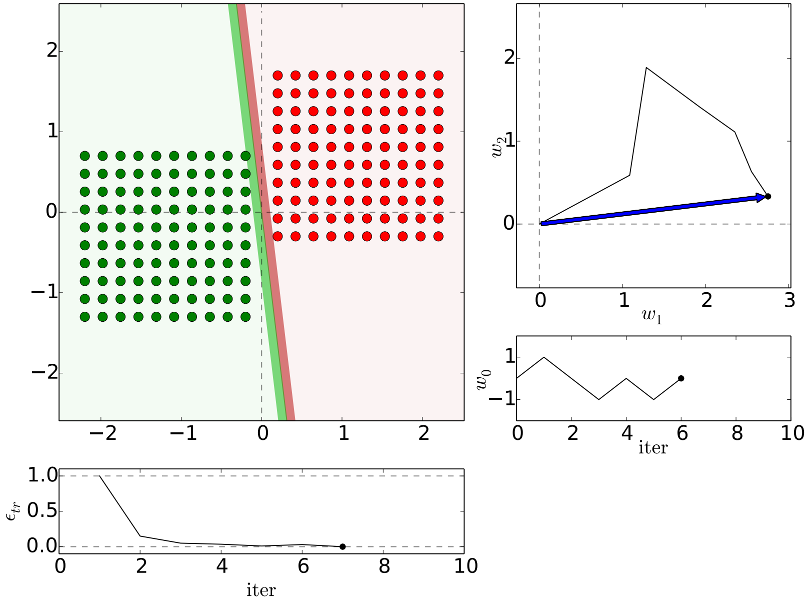
Perceptron iterations



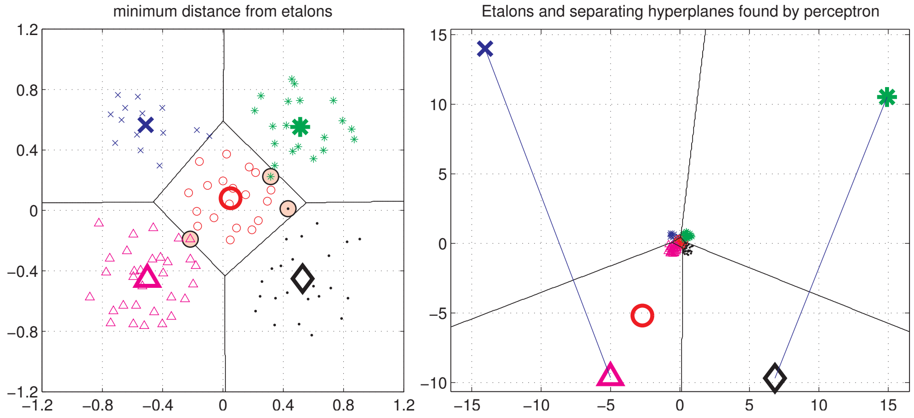
Perceptron iterations



Perceptron iterations

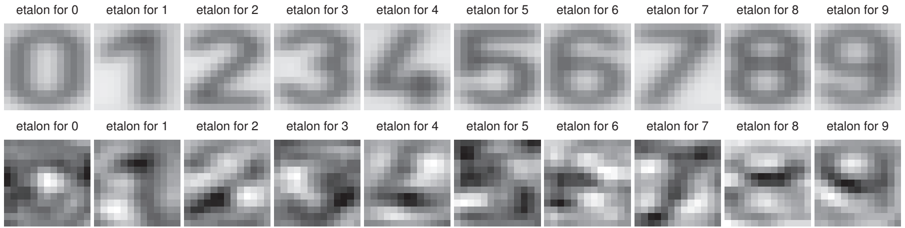


Etalons: means vs found be perceptron



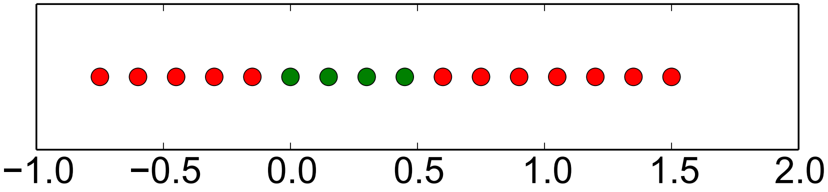
Figures from [5]

Digit recognition - etalons means vs. perceptron



Figures from [5]

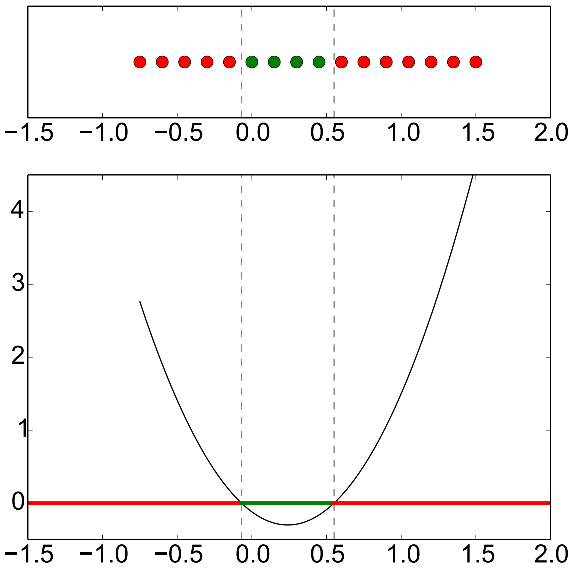
What if not lin separable?



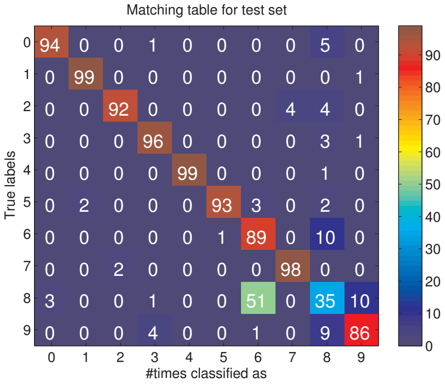
Dimension lifting

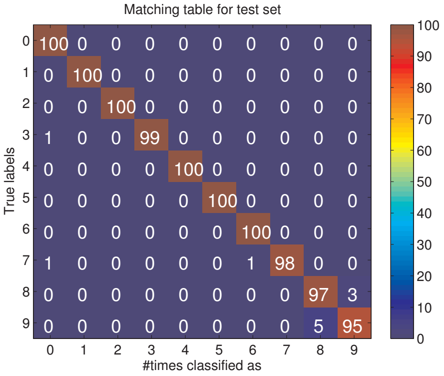
$$\mathbf{x} = [x, x^2]^\top$$

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



Performance comparison, parameters fixed





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

Precision and Recall, Confusion matrix

Consider digit detection (is there a digit?) or SPAM/HAM classification.

Confusion matrix :

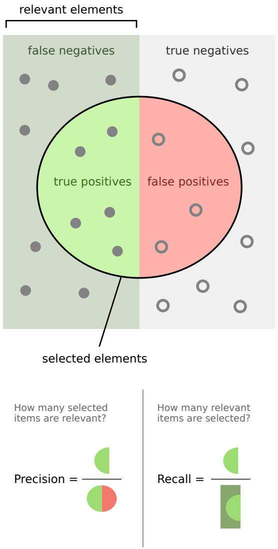
- Classification (prediction) vs Truth state

Recall :

- How many relevant items are selected?
- Are we missing some items?
- Also called: True positive rate, sensitivity, hit rate ...

Precision

- How many selected items are relevant?
- Also called: Positive predictive value



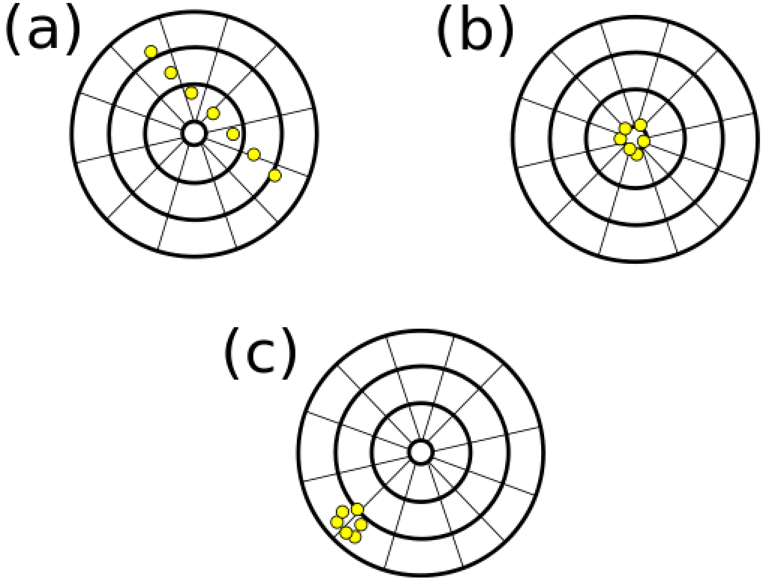
By Walber - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=36926283

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Think about precision vs recall graph, what is the best classifier?

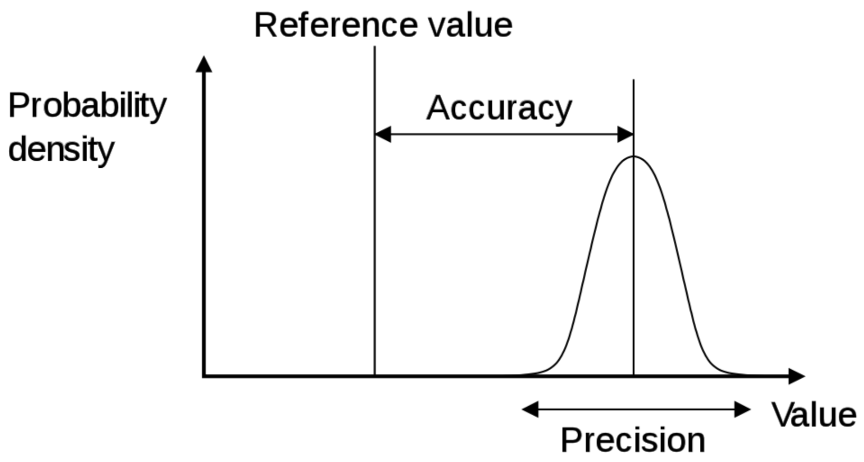
Accuracy vs precision



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

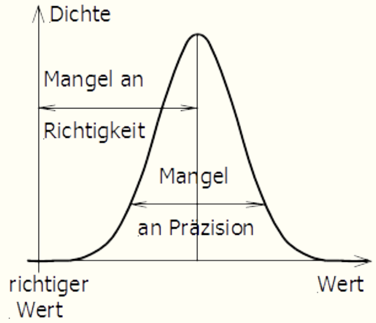
Accuracy: how close (is your model) to the true. Precision: how consistent/stable

Accuracy vs precision



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

Accuracy: how close (is your model) to the true. Precision: how consistent/stable.
Think about terms *bias* and *error*. In Czech perhaps accuracy≈správnost, precision≈přesnost.



References I

Further reading: Chapter 13 and 14 of [4]. Books [1] and [2] are classical textbooks in the field of pattern recognition and machine learning. Many Matlab figures created with the help of [3]

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

[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`.

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/.