Linear Models for Regression and Classification, Learning

Tomáš Svoboda and Petr Pošík
thanks to Matěj Hoffmann, Daniel Novák, 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 18, 2024

Contents

Supervised learning

A training multi-set of examples is available. Correct answers (hidden state, class, the quantity we want to predict) are *known* for all training examples.

Classification :

- Nominal dependent variable
- Examples: predict spam/ham based on email contents, predict 0/1/.../9 based on the image of a number, etc.

Regression :

- Quantitative/continuous dependent variable
- Examples: predict temperature in Prague based on date and time, predict height of a person based on weight and gender, etc.

Supervised learning

A training multi-set of examples is available. Correct answers (hidden state, class, the quantity we want to predict) are *known* for all training examples.

Classification :

- Nominal dependent variable
- Examples: predict spam/ham based on email contents, predict $0/1/\ldots/9$ based on the image of a number, etc.

Regression :

- Quantitative/continuous dependent variable
- Examples: predict temperature in Prague based on date and time, predict height of a person based on weight and gender, etc.

Supervised learning

A training multi-set of examples is available. Correct answers (hidden state, class, the quantity we want to predict) are *known* for all training examples.

Classification :

- Nominal dependent variable
- Examples: predict spam/ham based on email contents, predict $0/1/\ldots/9$ based on the image of a number, etc.

Regression :

- Quantitative/continuous dependent variable
- Examples: predict temperature in Prague based on date and time, predict height of a person based on weight and gender, etc.

Learning: minimization of empirical risk

- Given the set of parametrized strategies $\delta\colon \mathcal{X} \to \mathcal{D}$, penalty/loss function $\ell\colon \mathcal{S} \times \mathcal{D} \to \mathbb{R}$, the quality of each strategy δ could be described by the risk

$$R(\delta) = \sum_{s \in \mathcal{S}} \sum_{x \in \mathcal{X}} P(x, s)\ell(s, \delta(x)),$$

  but P is unknown.

- We thus use the empirical risk R_{emp}, i.e., average loss on training (multi)set $\mathcal{T} = \{(x^{(i)}, s^{(i)})\}_{i=1}^{N}$, $x \in \mathcal{X}$, $s \in \mathcal{S}$:

$$R_{\mathrm{emp}}(\delta) = \frac{1}{N} \sum_{(x^{(i)}, s^{(i)}) \in \mathcal{T}} \ell(s^{(i)}, \delta(x^{(i)})).$$

- Optimal strategy $\delta^* = \operatorname{argmin}_{\delta} R_{\mathrm{emp}}(\delta)$.
- We assume data $\mathcal{T}$ are from distribution $P(x, s)$.

Learning: minimization of empirical risk

- Given the set of parametrized strategies $\delta\colon \mathcal{X} \to \mathcal{D}$, penalty/loss function $\ell\colon \mathcal{S} \times \mathcal{D} \to \mathbb{R}$, the quality of each strategy δ could be described by the risk

$$R(\delta) = \sum_{s \in \mathcal{S}} \sum_{x \in \mathcal{X}} P(x, s) \ell(s, \delta(x)),$$

but P is unknown.

- We thus use the empirical risk R_{emp}, i.e., average loss on training (multi)set $\mathcal{T} = \{(x^{(i)}, s^{(i)})\}_{i=1}^{N}$, $x \in \mathcal{X}$, $s \in \mathcal{S}$:

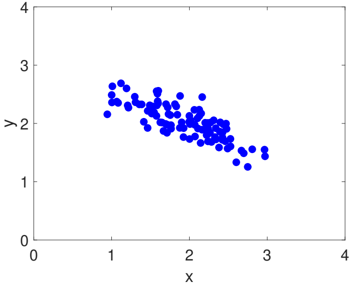
$$R_{\mathrm{emp}}(\delta) = \frac{1}{N} \sum_{(x^{(i)}, s^{(i)}) \in \mathcal{T}} \ell(s^{(i)}, \delta(x^{(i)})).$$

- Optimal strategy $\delta^* = \operatorname{argmin}_\delta R_{\mathrm{emp}}(\delta)$.
- We assume data $\mathcal{T}$ are from distribution $P(x, s)$.

Contents

Quiz: Line fitting

We would like to fit a line of the form $\widehat{y} = w_0 + w_1 x$ to the following data:



The parameters of a line with the best fit will likely be

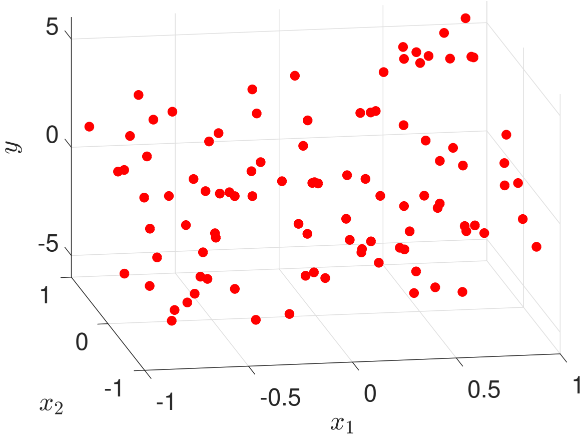
A $w_0 = -1,\ w_1 = -2$

B $w_0 = -\frac{1}{2},\ w_1 = 1$

C $w_0 = 3,\ w_1 = -\frac{1}{2}$

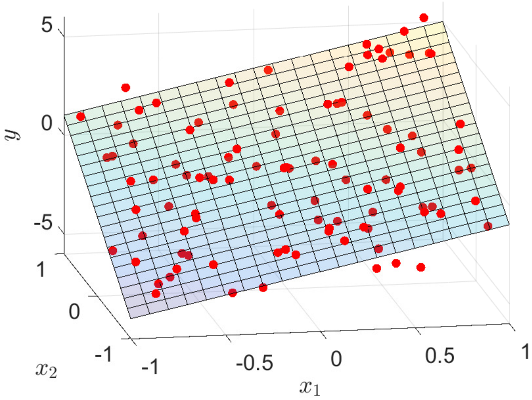
D $w_0 = 2,\ w_1 = \frac{1}{3}$

Linear regression: Illustration



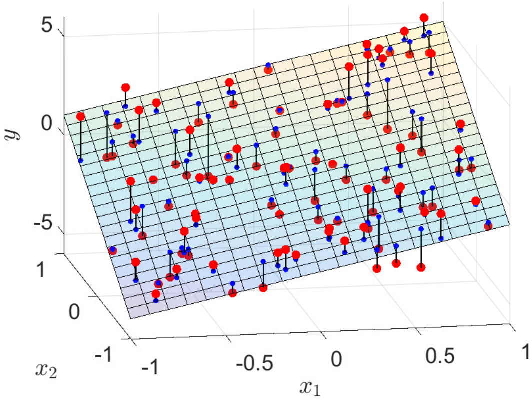
Given a dataset of input vectors $\vec{x}^{(i)}$ and the respective values of output variable $y^{(i)}$ ...

Linear regression: Illustration



. . . we would like to find a linear model of this dataset . . .

Linear regression: Illustration



... minimizing the errors between target values and the model predictions.

Regression

Reformulating Linear algebra in a machine learning language.

Regression task is a supervised learning task, i.e.

- a training (multi)set $\mathcal{T} = \{(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(N)}, y^{(N)})\}$ is available, where
- the labels $y^{(i)}$ are *quantitative*, often *continuous* (as opposed to classification tasks where $y^{(i)}$ are nominal).
- Its purpose is to model the relationship between independent variables (inputs) $\vec{x} = (x_1, \ldots, x_D)$ and the dependent variable (output) y.

Linear Regression

Linear regression uses a particular regression model which assumes (and learns) linear relationship between the inputs and the output:

$$\widehat{y} = \delta(\vec{x}) = w_0 + w_1 x_1 + \ldots + w_D x_D = w_0 + \langle \vec{w}, \vec{x} \rangle = w_0 + \vec{w}^\top \vec{x},$$

where

- $\widehat{y}$ is the model *prediction* (*estimate* of the true value y),
- $\delta(\vec{x})$ is the decision strategy (a linear model in this case),
- $w_0, \ldots, w_D$ are the coefficients of the linear function (weights), w_0 is the *bias*,
- $\langle \vec{w}, \vec{x} \rangle$ is a *dot product* of vectors $\vec{w}$ and $\vec{x}$ (scalar product),
- which can be also computed as a matrix product $\vec{w}^\top \vec{x}$ if $\vec{w}$ and $\vec{x}$ are *column vectors*, i.e. matrices of size $[D \times 1]$.

Notation remarks

Homogeneous coordinates :

- If we add "1" as the first element of $\vec{x}$ so that $\vec{x} = (1, x_1, \ldots, x_D)$, and
- if we include the bias term w_0 in the vector $\vec{w}$ so that $\vec{w} = (w_0, w_1, \ldots, w_D)$, then

$$\widehat{y} = \delta(\vec{x}) = w_0 \cdot 1 + w_1 x_1 + \ldots + w_D x_D = \langle \vec{w}, \vec{x} \rangle = \vec{w}^\top \vec{x}.$$

Matrix notation: If we organize the data $\mathcal{T}$ into matrices $\mathbf{X}$ and $\mathbf{Y}$, such that

$$\mathbf{X} = \begin{pmatrix} 1 & \ldots & 1 \\ \vec{x}^{(1)} & \ldots & \vec{x}^{(N)} \end{pmatrix} \quad \text{and} \quad \mathbf{Y} = \left(y^{(1)}, \ldots, y^{(N)}\right),$$

then we can write a batch computation of predictions for all data in $\mathbf{X}$ as

$$\widehat{\mathbf{Y}} = \left(\delta(\vec{x}^{(1)}), \ldots, \delta(\vec{x}^{(N)})\right) = \left(\vec{w}^\top \vec{x}^{(1)}, \ldots, \vec{w}^\top \vec{x}^{(N)}\right) = \vec{w}^\top \mathbf{X}.$$

Notation remarks

Homogeneous coordinates :

- If we add "1" as the first element of $\vec{x}$ so that $\vec{x} = (1, x_1, \ldots, x_D)$, and
- if we include the bias term w_0 in the vector $\vec{w}$ so that $\vec{w} = (w_0, w_1, \ldots, w_D)$, then

$$\widehat{y} = \delta(\vec{x}) = w_0 \cdot 1 + w_1 x_1 + \ldots + w_D x_D = \langle \vec{w}, \vec{x} \rangle = \vec{w}^\top \vec{x}.$$

Matrix notation: If we organize the data $\mathcal{T}$ into matrices $\mathbf{X}$ and $\mathbf{Y}$, such that

$$\mathbf{X} = \begin{pmatrix} 1 & \ldots & 1 \\ \vec{x}^{(1)} & \ldots & \vec{x}^{(N)} \end{pmatrix} \qquad \text{and} \qquad \mathbf{Y} = \left(y^{(1)}, \ldots, y^{(N)} \right),$$

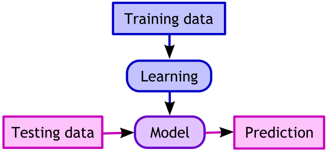
then we can write a batch computation of predictions for all data in $\mathbf{X}$ as

$$\widehat{\mathbf{Y}} = \left(\delta(\vec{x}^{(1)}), \ldots, \delta(\vec{x}^{(N)}) \right) = \left(\vec{w}^\top \vec{x}^{(1)}, \ldots, \vec{w}^\top \vec{x}^{(N)} \right) = \vec{w}^\top \mathbf{X}.$$

Two operation phases of ML models

Any ML model has 2 operation phases:

1. learning (training, fitting) of δ and
2. application of δ (testing, making predictions).



The strategy δ can be viewed as a function of 2 variables: $\delta(\vec{x}, \vec{w})$.

Model application (Inference): Given $\vec{w}$, we can manipulate $\vec{x}$ to make predictions:

$$\widehat{y} = \delta(\vec{x}, \vec{w}) = \delta_{\vec{w}}(\vec{x}).$$

Model learning: Given $\mathcal{T}$, we can tune the model parameters $\vec{w}$ to fit the model to the data:

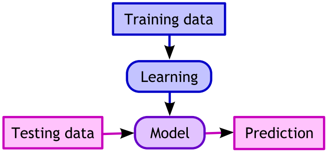
$$\vec{w}^* = \underset{\vec{w}}{\operatorname{argmin}}\, R_{\text{emp}}(\delta_{\vec{w}}) = \underset{\vec{w}}{\operatorname{argmin}}\, J(\vec{w}, \mathcal{T}),$$

where usually $J(\vec{w}, \mathcal{T}) = \dfrac{1}{|\mathcal{T}|} \sum_{(\vec{x},y)\in\mathcal{T}} \ell(y, \delta(\vec{x}, \vec{w}))$. How to train the model?

Two operation phases of ML models

Any ML model has 2 operation phases:

1. learning (training, fitting) of δ and
2. application of δ (testing, making predictions).



The strategy δ can be viewed as a function of 2 variables: $\delta(\vec{x}, \vec{w})$.

Model application (Inference): Given $\vec{w}$, we can manipulate $\vec{x}$ to make predictions:

$$\hat{y} = \delta(\vec{x}, \vec{w}) = \delta_{\vec{w}}(\vec{x}).$$

Model learning: Given $\mathcal{T}$, we can tune the model parameters $\vec{w}$ to fit the model to the data:

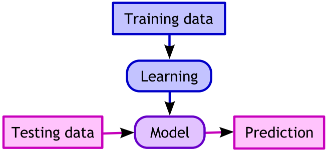
$$\vec{w}^* = \operatorname*{argmin}_{\vec{w}} R_{\text{emp}}(\delta_{\vec{w}}) = \operatorname*{argmin}_{\vec{w}} J(\vec{w}, \mathcal{T}),$$

where usually $J(\vec{w}, \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{(\vec{x},y)\in\mathcal{T}} \ell(y, \delta(\vec{x}, \vec{w}))$. How to train the model?

Two operation phases of ML models

Any ML model has 2 operation phases:

1. learning (training, fitting) of δ and
2. application of δ (testing, making predictions).



The strategy δ can be viewed as a function of 2 variables: $\delta(\vec{x}, \vec{w})$.

Model application (Inference): Given $\vec{w}$, we can manipulate $\vec{x}$ to make predictions:

$$\widehat{y} = \delta(\vec{x}, \vec{w}) = \delta_{\vec{w}}(\vec{x}).$$

Model learning: Given $\mathcal{T}$, we can tune the model parameters $\vec{w}$ to fit the model to the data:

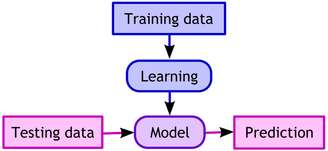
$$\vec{w}^* = \operatorname*{argmin}_{\vec{w}} R_{\mathrm{emp}}(\delta_{\vec{w}}) = \operatorname*{argmin}_{\vec{w}} J(\vec{w}, \mathcal{T}),$$

where usually $J(\vec{w}, \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{(\vec{x}, y) \in \mathcal{T}} \ell(y, \delta(\vec{x}, \vec{w}))$. How to train the model?

Two operation phases of ML models

Any ML model has 2 operation phases:

1. learning (training, fitting) of δ and
2. application of δ (testing, making predictions).



The strategy δ can be viewed as a function of 2 variables: $\delta(\vec{x}, \vec{w})$.

Model application (Inference): Given $\vec{w}$, we can manipulate $\vec{x}$ to make predictions:

$$\widehat{y} = \delta(\vec{x}, \vec{w}) = \delta_{\vec{w}}(\vec{x}).$$

Model learning: Given $\mathcal{T}$, we can tune the model parameters $\vec{w}$ to fit the model to the data:

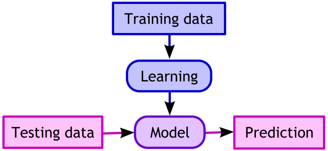
$$\vec{w}^* = \operatorname*{argmin}_{\vec{w}} R_{\mathrm{emp}}(\delta_{\vec{w}}) = \operatorname*{argmin}_{\vec{w}} J(\vec{w}, \mathcal{T}),$$

where usually $J(\vec{w}, \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{(\vec{x},y)\in\mathcal{T}} \ell(y, \delta(\vec{x}, \vec{w}))$. How to train the model?

Two operation phases of ML models

Any ML model has 2 operation phases:

1. learning (training, fitting) of δ and
2. application of δ (testing, making predictions).



The strategy δ can be viewed as a function of 2 variables: $\delta(\vec{x}, \vec{w})$.

Model application (Inference): Given $\vec{w}$, we can manipulate $\vec{x}$ to make predictions:

$$\widehat{y} = \delta(\vec{x}, \vec{w}) = \delta_{\vec{w}}(\vec{x}).$$

Model learning: Given $\mathcal{T}$, we can tune the model parameters $\vec{w}$ to fit the model to the data:

$$\vec{w}^* = \underset{\vec{w}}{\operatorname{argmin}}\, R_{\text{emp}}(\delta_{\vec{w}}) = \underset{\vec{w}}{\operatorname{argmin}}\, J(\vec{w}, \mathcal{T}),$$

where usually $J(\vec{w}, \mathcal{T}) = \dfrac{1}{|\mathcal{T}|} \displaystyle\sum_{(\vec{x},y)\in\mathcal{T}} \ell(y, \delta(\vec{x}, \vec{w}))$. How to train the model?

Example: Simple (univariate) linear regression

Simple regression

- $\vec{x}^{(i)} = x^{(i)}$, i.e., the examples are described by a single feature (they are 1-dimensional).
- Find parameters w_0, w_1 of a linear model $\hat{y} = w_0 + w_1 x$
  given a training (multi)set $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$.

How many lines can be fit to N linearly independent training examples?

- $N = 1$ (1 equation, 2 parameters) $\Rightarrow \infty$ linear functions with zero error
- $N = 2$ (2 equation, 2 parameters) $\Rightarrow$ 1 linear function with zero error
- $N \geq 3$ (> 2 equation, parameters) $\Rightarrow$ no linear function with zero error
  $\Rightarrow$ but we can fit a line which minimizes the "size" of error $y - \widehat{y}$:

$$\vec{w}^* = (w_0^*, w_1^*) = \operatorname*{argmin}_{w_0, w_1} R_{\mathrm{emp}}(w_0, w_1) = \operatorname*{argmin}_{w_0, w_1} J(w_0, w_1, \mathcal{T}).$$

Example: Simple (univariate) linear regression

Simple regression

- $\vec{x}^{(i)} = x^{(i)}$, i.e., the examples are described by a single feature (they are 1-dimensional).
- Find parameters w_0, w_1 of a linear model $\hat{y} = w_0 + w_1 x$
  given a training (multi)set $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$.

How many lines can be fit to N linearly independent training examples?

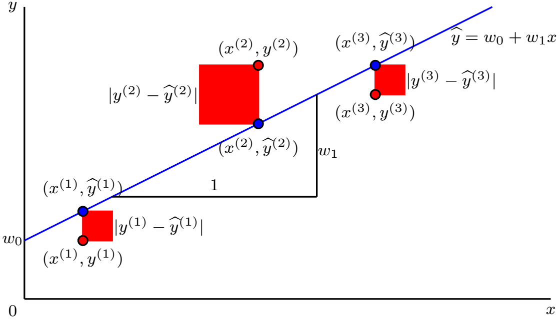
- $N = 1$ (1 equation, 2 parameters) $\Rightarrow$ ∞ linear functions with zero error
- $N = 2$ (2 equation, 2 parameters) $\Rightarrow$ 1 linear function with zero error
- $N \geq 3$ ($>$ 2 equation, parameters) $\Rightarrow$ no linear function with zero error
  $\Rightarrow$ but we can fit a line which minimizes the "size" of error $y - \widehat{y}$:

$$\vec{w}^* = (w_0^*, w_1^*) = \operatorname*{argmin}_{w_0, w_1} R_{\text{emp}}(w_0, w_1) = \operatorname*{argmin}_{w_0, w_1} J(w_0, w_1, \mathcal{T}).$$

The least squares method

Choose such parameters $\vec{w}$ which minimize the *mean squared error* (MSE)

$$J_{MSE}(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \widehat{y}^{(i)}\right)^2$$
$$= \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \delta_{\vec{w}}(\vec{x}^{(i)})\right)^2 .$$



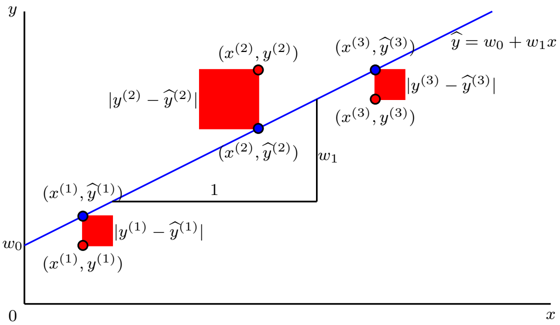
Is there a (closed-form) solution? Explicit solution:

$$w_1 = \frac{\sum_{i=1}^{N}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_{i=1}^{N}(x^{(i)} - \bar{x})^2} = \frac{s_{xy}}{s_x^2} = \frac{\text{covariance of } X \text{ and } Y}{\text{variance } X} \qquad w_0 = \bar{y} - w_1\bar{x}$$

The least squares method

Choose such parameters $\vec{w}$ which minimize the *mean squared error* (MSE)

$$J_{MSE}(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \widehat{y}^{(i)}\right)^2$$
$$= \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \delta_{\vec{w}}(\vec{x}^{(i)})\right)^2 .$$

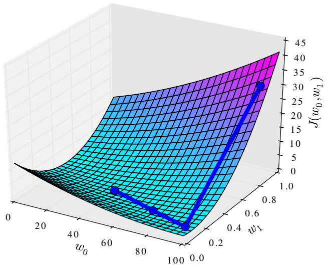


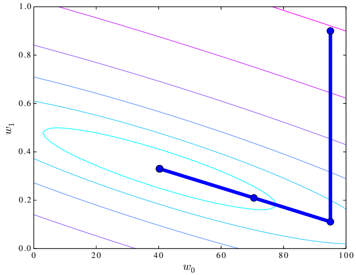
Is there a (closed-form) solution? Explicit solution:

$$w_1 = \frac{\sum_{i=1}^{N}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_{i=1}^{N}(x^{(i)} - \bar{x})^2} = \frac{s_{xy}}{s_x^2} = \frac{\text{covariance of } X \text{ and } Y}{\text{variance } X} \qquad w_0 = \bar{y} - w_1\bar{x}$$

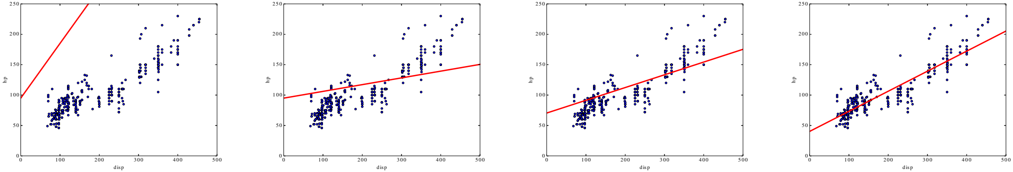
Universal fitting method: minimization of cost function J

The landscape of J in the space of parameters w_0 and w_1 (for the data below):





Gradually better linear models found by an optimization method (BFGS):



Gradient descent algorithm

Given a function $J(w_0, w_1)$ that should be minimized,

- start with a guess of w_0 and w_1 and
- change it, so that $J(w_0, w_1)$ decreases, i.e.
- update our current guess of w_0 and w_1 by taking a step in the direction opposite to the gradient:

$$\vec{w} \leftarrow \vec{w} - \alpha \nabla J(w_0, w_1), \text{ i.e.}$$
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} J(w_0, w_1),$$

where all w_is are updated simultaneously and α is a learning rate (step size).

Gradient descent for MSE minimization

For the cost function

$$J(w_0, w_1) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \delta_{\vec{w}}(x^{(i)}) \right)^2 = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - (w_0 + w_1 x^{(i)}) \right)^2 ,$$

the gradient can be computed as

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = -\frac{2}{N} \sum_{i=1}^{N} \left(y^{(i)} - \delta_{\vec{w}}(x^{(i)}) \right)$$

$$\frac{\partial}{\partial w_1} J(w_0, w_1) = -\frac{2}{N} \sum_{i=1}^{N} \left(y^{(i)} - \delta_{\vec{w}}(x^{(i)}) \right) x^{(i)}$$

Multivariate linear regression

- $\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_D^{(i)})^\top$, i.e. the examples are described by more than 1 feature (they are D-dimensional).
- Find the parameters $\vec{w} = (w_0, \dots, w_D)^\top$ of a linear model $\widehat{y} = \vec{w}^\top \vec{x}$ given the training (multi)set $\mathcal{T} = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.

Training: we would like
for each (i): $y^{(i)} = \vec{w}^\top \vec{x}^{(i)}$.
Or, in the matrix form: $\mathbf{Y} = \vec{w}^\top \mathbf{X}$

What is the shape of $\mathbf{X}$?

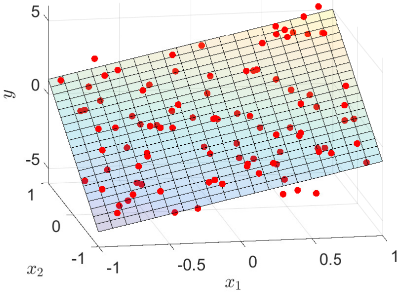
A $(D+1) \times (D+1)$

B $(D+1) \times N$

C $N \times (D+1)$

D $N \times N$

The model is a *hyperplane* in the $(D+1)$-dimensional space.

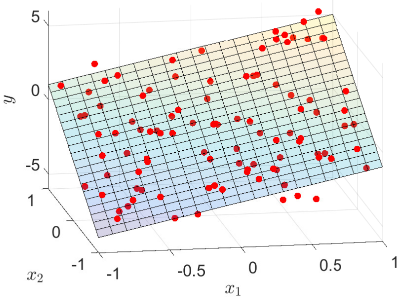


Multivariate linear regression

- $\vec{x}^{(i)} = (x_1^{(i)}, \ldots, x_D^{(i)})^\top$, i.e. the examples are described by more than 1 feature (they are D-dimensional).
- Find the parameters $\vec{w} = (w_0, \ldots, w_D)^\top$ of a linear model $\widehat{y} = \vec{w}^\top \vec{x}$ given the training (multi)set $\mathcal{T} = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.

Training: we would like
for each (i): $y^{(i)} = \vec{w}^\top \vec{x}^{(i)}$.
Or, in the matrix form: $\mathbf{Y} = \vec{w}^\top \mathbf{X}$

The model is a *hyperplane*
in the $(D+1)$-dimensional space.



What is the shape of $\mathbf{X}$?

A $(D+1) \times (D+1)$

B $(D+1) \times N$

C $N \times (D+1)$

D $N \times N$

Multivariate linear regression: learning

1. Numeric optimization of $J(\vec{w}, T)$:
   - Works as for simple regression, it only searches a space with more dimensions.
   - Sometimes one needs to tune some parameters of the optimization algorithm to work properly (learning rate in gradient descent, etc.).
   - May be slow (many iterations needed), but works even for very large D.

2. Normal equation :

$$\vec{w}^* = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y}^\top$$

   - Method to solve for the optimal $\vec{w}^*$ analytically!
   - No need to choose optimization algorithm parameters. No iterations.
   - Needs to compute $(\mathbf{X}\mathbf{X}^\top)^{-1}$, which is $O((D+1)^3)$. Becomes intractable for large D.

Multivariate linear regression: learning

1. Numeric optimization of $J(\vec{w}, T)$:
   - Works as for simple regression, it only searches a space with more dimensions.
   - Sometimes one needs to tune some parameters of the optimization algorithm to work properly (learning rate in gradient descent, etc.).
   - May be slow (many iterations needed), but works even for very large D.
2. Normal equation :

$$\vec{w}^* = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y}^\top$$

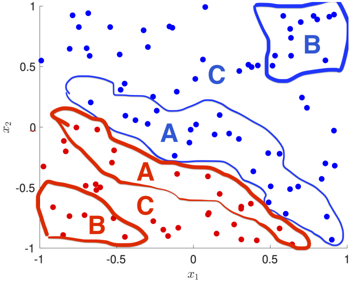
   - Method to solve for the optimal $\vec{w}^*$ analytically!
   - No need to choose optimization algorithm parameters. No iterations.
   - Needs to compute $(\mathbf{X}\mathbf{X}^\top)^{-1}$, which is $O((D+1)^3)$. Becomes intractable for large D.

Contents

Classification

- Binary classification
- Discriminant function
- Classification as a regression problem (linear, logistic regression)
- What is the right loss function?
- Etalon classifier (meeting nearest neighbour and linear classifier)
- Acuracy vs precision

Quiz: Importance of training examples



Intuitively, which of the training data points should have the biggest influence on the decision whether a new, unlabeled data point shall be red or blue?

- **A** Those which are closest to data points with the opposite color.
- **B** Those which are farthest from the data points of the opposite color.
- **C** Those which are near the middle of the points with the same color.
- **D** None. All of the data points have the same importance.

Binary classification task

Let's have a training dataset $\mathcal{T} = \{(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(N)}, y^{(N)})$:

- each example described by a vector $\vec{x} = (x_1, \ldots, x_D)$,
- labeled with the correct class $y \in \{+1, -1\}$.

The goal:

- Find the classifier (decision strategy/rule) δ that minimizes the empirical risk $R_{\mathrm{emp}}(\delta)$.

Discriminant function

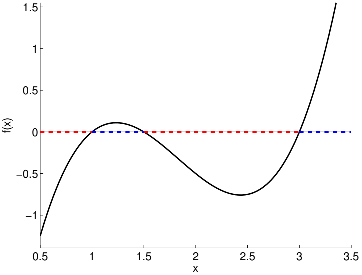
Discriminant function $f(\vec{x})$:

- It assigns a real number to each observation $\vec{x}$. It may be linear or non-linear.
- For 2 classes, 1 discriminant function is enough.
- It is used to create a **decision rule** (which then assigns a class to an observation):

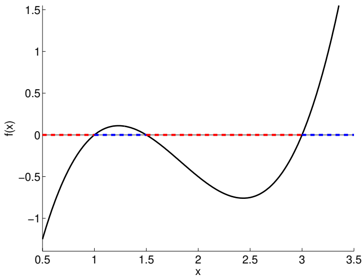
$$\widehat{y} = \delta(\vec{x}) = \begin{cases} +1 & \text{iff} \quad f(\vec{x}) > 0, \text{ and} \\ -1 & \text{iff} \quad f(\vec{x}) < 0, \end{cases}$$

i.e., $\widehat{y} = \delta(\vec{x}) = \operatorname{sign}(f(\vec{x}))$.

- Decision boundary: $\{\vec{x} | f(\vec{x}) = 0\}$
- Linear classification: **the decision boundaries must be linear.**
- *Learning* then amounts to finding a suitable function f (or its parameters).



Discriminant function



Discriminant function $f(\vec{x})$:

- It assigns a real number to each observation $\vec{x}$. It may be linear or non-linear.
- For 2 classes, 1 discriminant function is enough.
- It is used to create a **decision rule** (which then assigns a class to an observation):

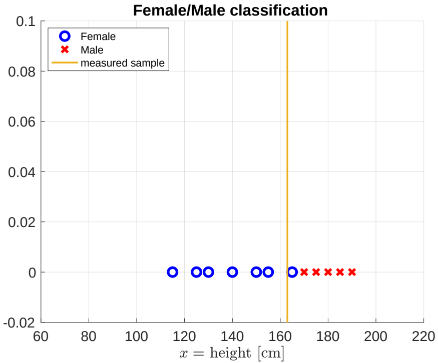
$$\widehat{y} = \delta(\vec{x}) = \begin{cases} +1 & \text{iff} \quad f(\vec{x}) > 0, \text{ and} \\ -1 & \text{iff} \quad f(\vec{x}) < 0, \end{cases}$$

  i.e., $\widehat{y} = \delta(\vec{x}) = \mathrm{sign}\left(f(\vec{x})\right)$.
- **Decision boundary:** $\{\vec{x} | f(\vec{x}) = 0\}$
- **Linear classification:** **the decision boundaries must be linear.**
- *Learning* then amounts to finding a suitable function f (or its parameters).

Example: Female/Male classification based on height

Training (multi)set $\mathcal{T} = \{(x^{(i)}, s^{(i)})\}_{i=1}^{N}$, $x^{(i)} \in \mathcal{X}$, $s^{(i)} \in \mathcal{S} = \{F, M\}$

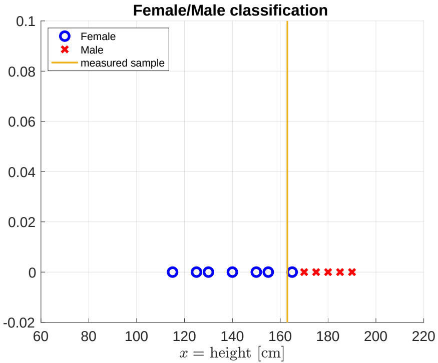
i	1	2	3	4	5	6	7	8	9	10	11	12
Height $x^{(i)}$	115	125	130	140	150	155	165	170	175	180	185	190
Gender $s^{(i)}$	F	F	F	F	F	F	F	M	M	M	M	M
Gender $y^{(i)}$ (+1/-1)	-1	-1	-1	-1	-1	-1	-1	+1	+1	+1	+1	+1



Example: Female/Male classification based on height

Training (multi)set $\mathcal{T} = \{(x^{(i)}, s^{(i)})\}_{i=1}^{N}$, $x^{(i)} \in \mathcal{X}$, $s^{(i)} \in \mathcal{S} = \{F, M\}$

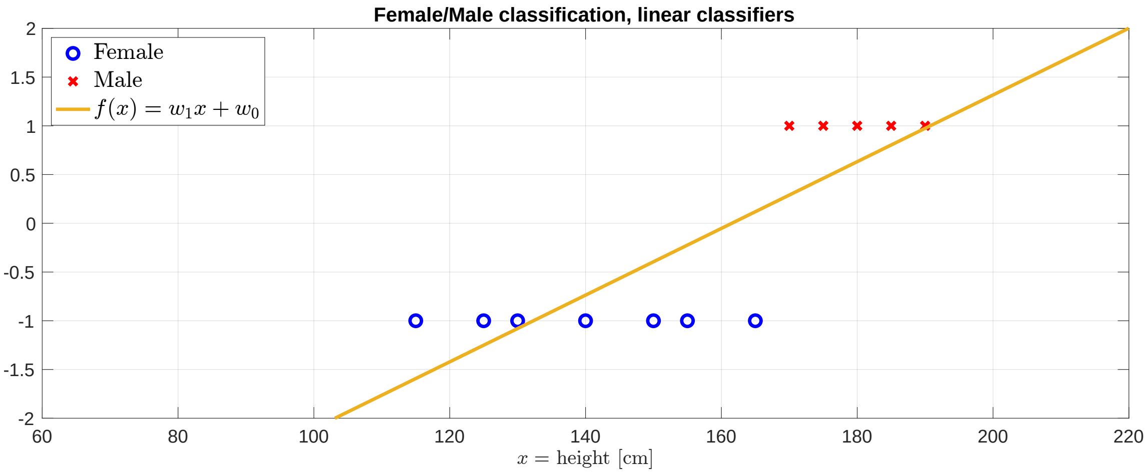
i	1	2	3	4	5	6	7	8	9	10	11	12
Height $x^{(i)}$	115	125	130	140	150	155	165	170	175	180	185	190
Gender $s^{(i)}$	F	F	F	F	F	F	F	M	M	M	M	M
Gender $y^{(i)}$ (+1/-1)	-1	-1	-1	-1	-1	-1	-1	+1	+1	+1	+1	+1



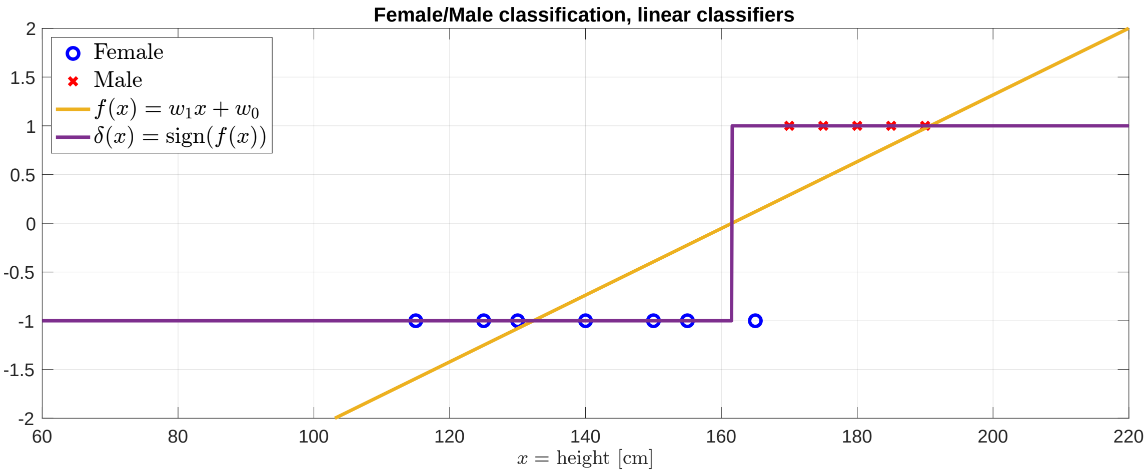
A new point to clasify: $x^Q = 163$

Which class does x^Q belong to? $\delta(x^Q) = ?$

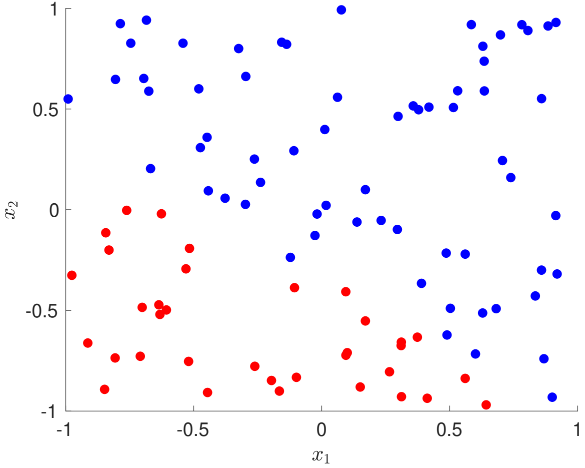
Example: Linear discr. function, LSQ fit



Example: Corresponding decision strategy

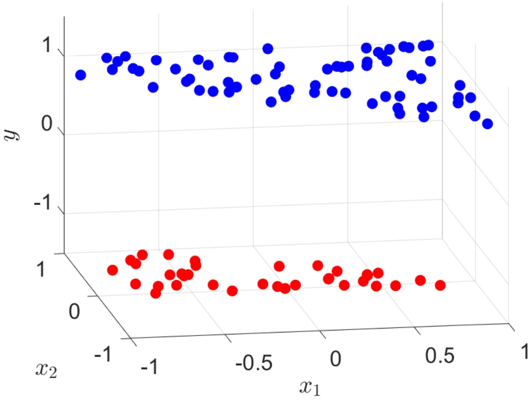


Learning linear classifier: naive approach



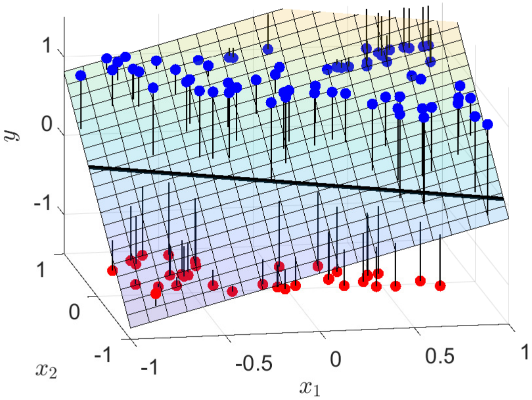
Let's have a dataset of input vectors $\vec{x}^{(i)}$ and their classes $s^{(i)}$.

Learning linear classifier: naive approach



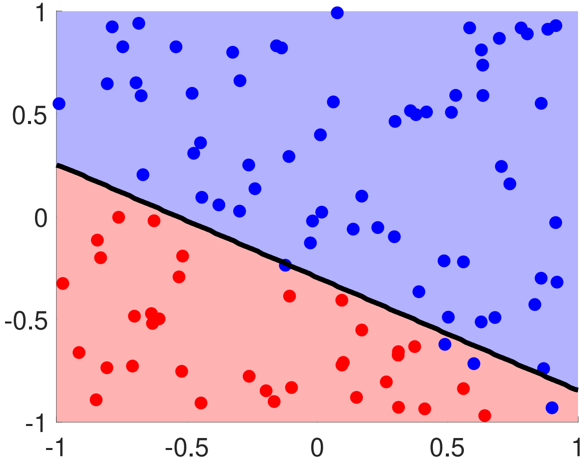
Let's encode the classes corresponding $y^{(i)} = -1$ or $y^{(i)} = 1$.

Learning linear classifier: naive approach



Let's fit a linear discriminant function by minimizing MSE as in regression.
The contour line $y = 0$ ...

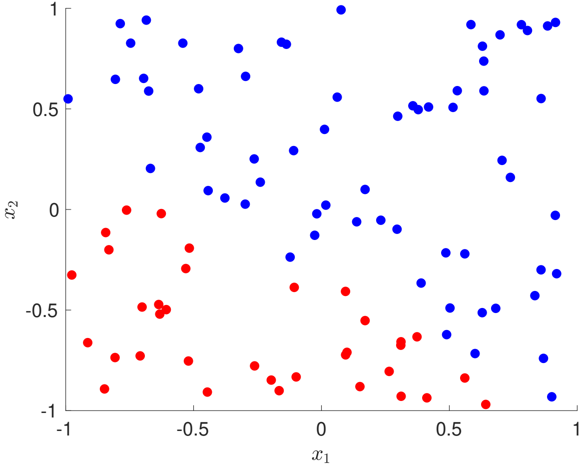
Learning linear classifier: naive approach



. . . then forms a linear decision boundary in the original 2D space.
But is such a classifier good in general?

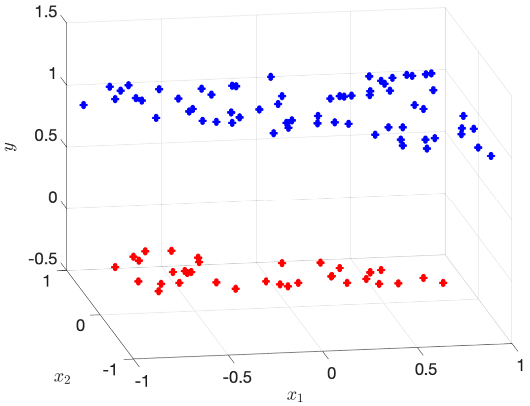
Can we do better than fitting a linear function?

Fitting a better function: Logistic regression



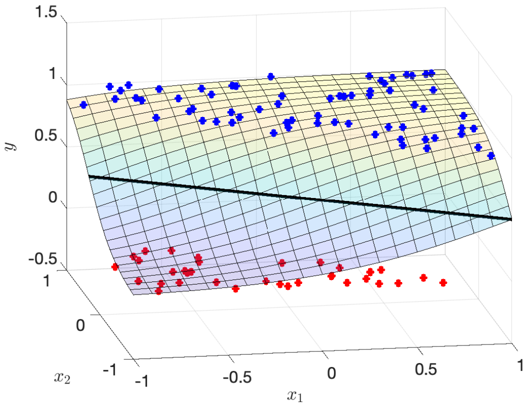
Let's have a dataset of input vectors $\vec{x}^{(i)}$ and their classes $s^{(i)}$.

Fitting a better function: Logistic regression



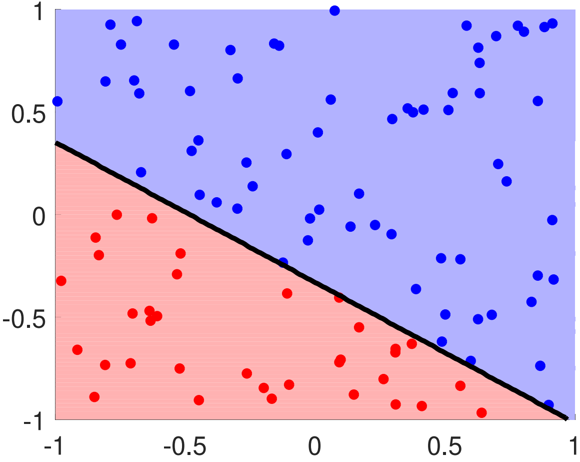
Let's encode the classes corresponding $y^{(i)} = 0$ or $y^{(i)} = 1$.

Fitting a better function: Logistic regression



Let's fit a **sigmoidal** discriminant function by minimizing MSE as in regression.
The contour line $y = 0.5$ ...

Fitting a better function: Logistic regression



. . . then forms a linear decision boundary in the original 2D space.

Logistic regression model

Logistic regression uses a discriminant function which is a nonlinear transformation of the values of a linear function

$$f_{\vec{w}}(\vec{x}) = g(\vec{w}^\top \vec{x}) = \frac{1}{1 + e^{-\vec{w}^\top \vec{x}}},$$

where $g(z) = \dfrac{1}{1 + e^{-z}}$ is the sigmoid function (a.k.a logistic function).

Interpretation of the model:

- $f_{\vec{w}}(\vec{x})$ can be interpretted as an estimate of the probability that $\vec{x}$ belongs to class 1.
- The decision boundary is defined using a level-set/countour $\{\vec{x} : f_{\vec{w}}(\vec{x}) = 0.5\}$.
- Logistic *regression* is a *classification* model!
- The discriminant function $f_{\vec{w}}(\vec{x})$ itself is not linear anymore; but the *decision boundary is still linear!*
- Thanks to the sigmoidal transformation, logistic regression is much less influenced by examples that are far from the decision boundary!

Logistic regression model

Logistic regression uses a discriminant function which is a nonlinear transformation of the values of a linear function

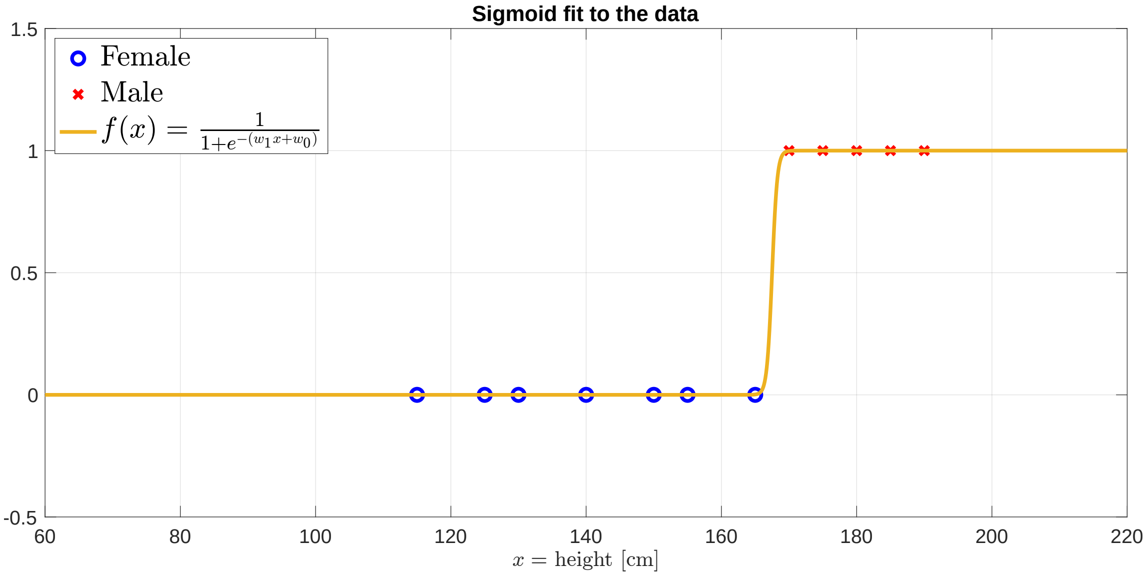
$$f_{\vec{w}}(\vec{x}) = g(\vec{w}^\top \vec{x}) = \frac{1}{1 + e^{-\vec{w}^\top \vec{x}}},$$

where $g(z) = \dfrac{1}{1 + e^{-z}}$ is the sigmoid function (a.k.a logistic function).

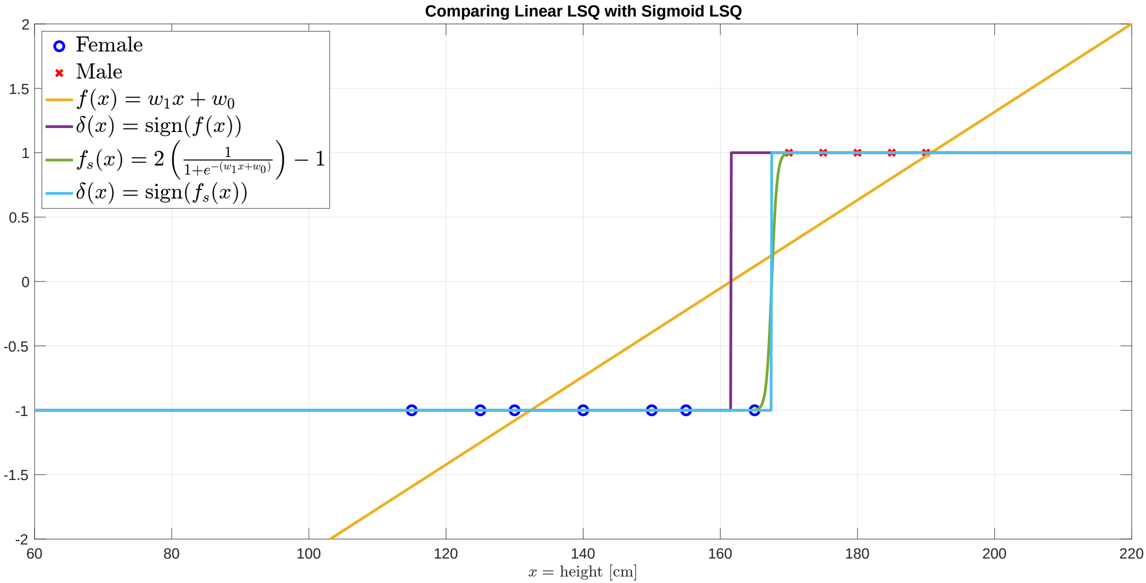
Interpretation of the model:

- $f_{\vec{w}}(\vec{x})$ can be interpretted as an estimate of the probability that $\vec{x}$ belongs to class 1.
- The decision boundary is defined using a level-set/countour $\{\vec{x} : f_{\vec{w}}(\vec{x}) = 0.5\}$.
- Logistic *regression* is a *classification* model!
- The discriminant function $f_{\vec{w}}(\vec{x})$ itself is not linear anymore; but the *decision boundary is still linear!*
- Thanks to the sigmoidal transformation, logistic regression is much less influenced by examples that are far from the decision boundary!

LSQ fit of a sigmoid



Comparing Linear and Sigmoid LSQ fit



What loss function ℓ is suitable?

To train the logistic regression model, one can minimize the J_{MSE} criterion:

- a non-convex, multimodal landscape which is hard to optimize.

Logistic regression uses a loss function called "cross-entropy":

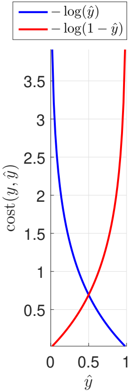
$$J(\vec{w}, \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} \ell(y^{(i)}, f_{\vec{w}}(\vec{x}^{(i)})), \text{ where}$$

$$\ell(y, \widehat{y}) = \begin{cases} -\log(\widehat{y}) & \text{if } y = 1 \\ -\log(1-\widehat{y}) & \text{if } y = 0 \end{cases},$$

which can be rewritten in a single expression as

$$\ell(y, \widehat{y}) = -y \cdot \log(\widehat{y}) - (1-y) \cdot \log(1-\widehat{y}).$$

- Easier to optimize for numerical solvers.



What loss function ℓ is suitable?

To train the logistic regression model, one can minimize the J_{MSE} criterion:

- a non-convex, multimodal landscape which is hard to optimize.

Logistic regression uses a loss function called cross-entropy :

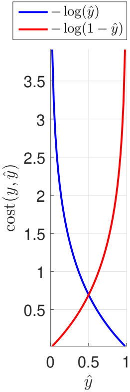
$$J(\vec{w}, \mathcal{T}) = \frac{1}{N} \sum_{i=1}^{N} \ell(y^{(i)}, f_{\vec{w}}(\vec{x}^{(i)})), \text{ where}$$

$$\ell(y, \widehat{y}) = \begin{cases} -\log(\widehat{y}) & \text{if } y = 1 \\ -\log(1 - \widehat{y}) & \text{if } y = 0 \end{cases},$$

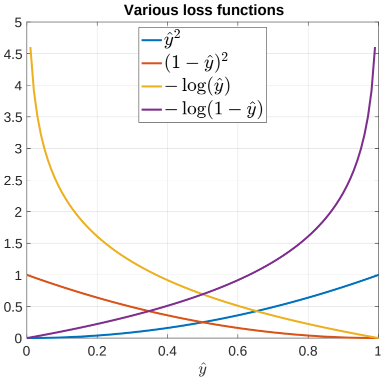
which can be rewritten in a single expression as

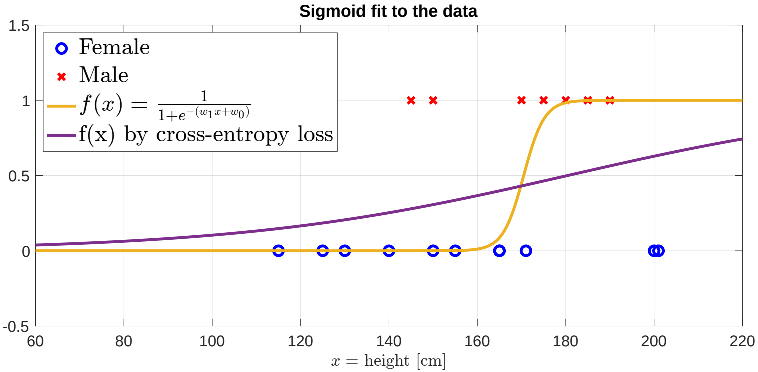
$$\ell(y, \widehat{y}) = -y \cdot \log(\widehat{y}) - (1 - y) \cdot \log(1 - \widehat{y}).$$

- Easier to optimize for numerical solvers.



MSE vs cross entropy loss



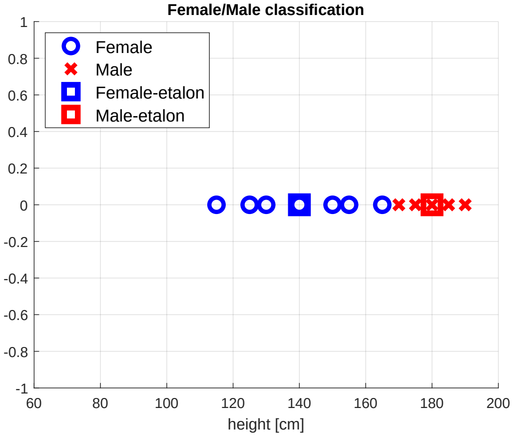


Sigmoidal $f(x)$ can be also interpreted as $P(s = \text{Male} \mid x)$: direct learning of a discriminative model .

Cross-entropy loss strongly penalizes hard errors, complete mismatches.

Alternative idea: Etalons

Represent each class by a single example called etalon ! (Or by a very small number of etalons.)



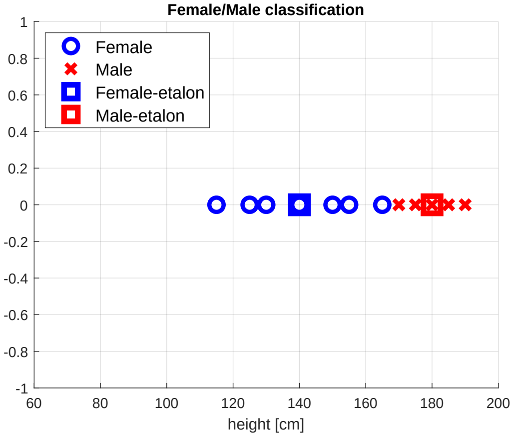
$$e_F = \mathrm{ave}(\{x^{(i)} : s^{(i)} = F\}) = 140$$
$$e_M = \mathrm{ave}(\{x^{(i)} : s^{(i)} = M\}) = 180$$

$x^Q = 163$

Based on etalons: $d^Q = \delta(x^Q) = ?$

Alternative idea: Etalons

Represent each class by a single example called etalon ! (Or by a very small number of etalons.)



$$e_F = \text{ave}(\{x^{(i)} : s^{(i)} = F\}) = 140$$
$$e_M = \text{ave}(\{x^{(i)} : s^{(i)} = M\}) = 180$$

$x^Q = 163$

Based on etalons: $d^Q = \delta(x^Q) = ?$

A $d^Q = F$

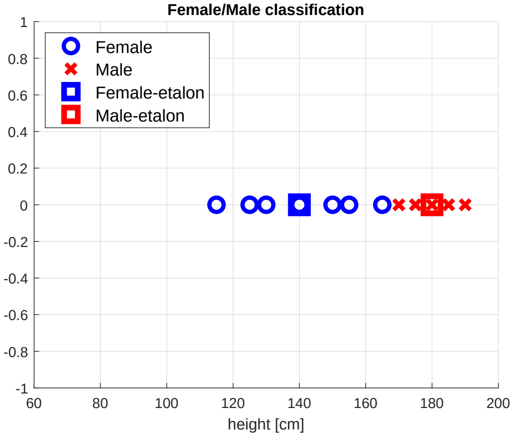
B $d^Q = M$

C Both classes equally likely

D Cannot provide any decision

Alternative idea: Etalons

Represent each class by a single example called etalon ! (Or by a very small number of etalons.)



$$e_F = \text{ave}(\{x^{(i)} : s^{(i)} = F\}) = 140$$
$$e_M = \text{ave}(\{x^{(i)} : s^{(i)} = M\}) = 180$$

$x^Q = 163$

Based on etalons: $d^Q = \delta(x^Q) = ?$

Classify as $d^Q = \text{argmin}_{s \in \mathcal{S}} \text{dist}(x^Q, e_s)$

What type of function is $\text{dist}(x^Q, e_s)$?

Etalon classifier is a Linear classifier!

Assuming $\operatorname{dist}(x, e) = (x - e)^2$, then

$$\operatorname*{argmin}_{s \in S} \operatorname{dist}(x, e_s) = \operatorname*{argmin}_{s \in S} (x - e_s)^2 = \operatorname*{argmin}_{s \in S} (\underbrace{x^2}_{\text{const.}} - 2 e_s x + e_s^2) =$$

$$= \operatorname*{argmin}_{s \in S} (-2 e_s x + e_s^2) = \operatorname*{argmax}_{s \in S} (\underbrace{e_s x - \frac{1}{2} e_s^2}_{\text{linear function of } x})$$

Multiclass classification: each class s has a linear discriminant function $f_s(x) = a_s x + b_s$ and

$$\delta(x) = \operatorname*{argmax}_{s \in S} f_s(x)$$

Binary classification: a single linear discriminant function $g(x)$ is sufficient and

$$\delta(x) = \begin{cases} s_1 & \text{if } g(x) \geq 0, \\ s_2 & \text{if } g(x) < 0. \end{cases}$$

Etalon classifier is a Linear classifier!

Assuming $\mathrm{dist}(x, e) = (x - e)^2$, then

$$\operatorname*{argmin}_{s \in S} \mathrm{dist}(x, e_s) = \operatorname*{argmin}_{s \in S} (x - e_s)^2 = \operatorname*{argmin}_{s \in S} (\underbrace{x^2}_{\text{const.}} - 2e_s x + e_s^2) =$$

$$= \operatorname*{argmin}_{s \in S} (-2e_s x + e_s^2) = \operatorname*{argmax}_{s \in S} (\underbrace{e_s x - \frac{1}{2} e_s^2}_{\text{linear function of } x})$$

Multiclass classification: each class s has a linear discriminant function $f_s(x) = a_s x + b_s$ and

$$\delta(x) = \operatorname*{argmax}_{s \in S} f_s(x)$$

Binary classification: a single linear discriminant function $g(x)$ is sufficient and

$$\delta(x) = \begin{cases} s_1 & \text{if } g(x) \geq 0, \\ s_2 & \text{if } g(x) < 0. \end{cases}$$

Etalon classifier is a Linear classifier!

Assuming $\mathrm{dist}(x, e) = (x - e)^2$, then

$$\operatorname*{argmin}_{s \in S} \mathrm{dist}(x, e_s) = \operatorname*{argmin}_{s \in S} (x - e_s)^2 = \operatorname*{argmin}_{s \in S} (\underbrace{x^2}_{\text{const.}} - 2e_s x + e_s^2) =$$

$$= \operatorname*{argmin}_{s \in S} (-2e_s x + e_s^2) = \operatorname*{argmax}_{s \in S} (\underbrace{e_s x - \frac{1}{2} e_s^2}_{\text{linear function of } x})$$

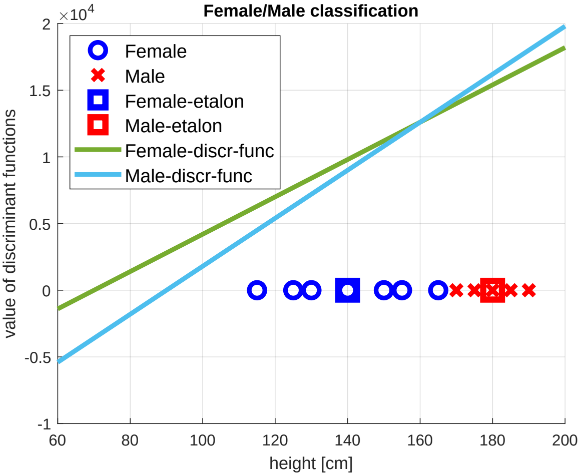
Multiclass classification: each class s has a linear discriminant function $f_s(x) = a_s x + b_s$ and

$$\delta(x) = \operatorname*{argmax}_{s \in S} f_s(x)$$

Binary classification: a single linear discriminant function $g(x)$ is sufficient and

$$\delta(x) = \begin{cases} s_1 & \text{if } g(x) \geq 0, \\ s_2 & \text{if } g(x) < 0. \end{cases}$$

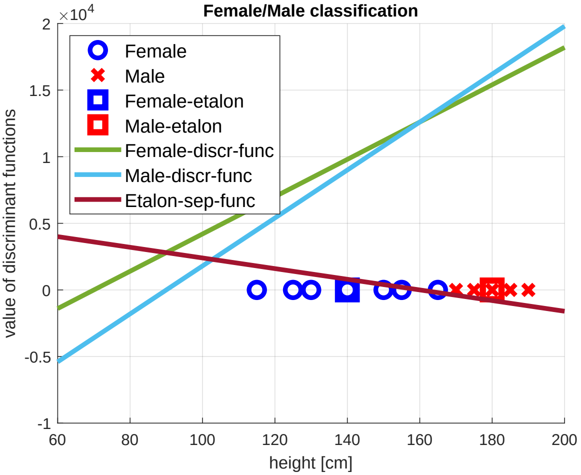
Example: F/M – Linear discriminant functions based on etalons



Discriminant functions for 2 classes:

$$
\begin{aligned}
f_F(x) &= a_F x + b_F = \\
&= e_F x - \frac{1}{2} e_F^2 = 140x - 9800 \\
f_M(x) &= a_M x + b_M = \\
&= e_M x - \frac{1}{2} e_M^2 = 180x - 16200
\end{aligned}
$$

Example: F/M – Linear discriminant functions based on etalons



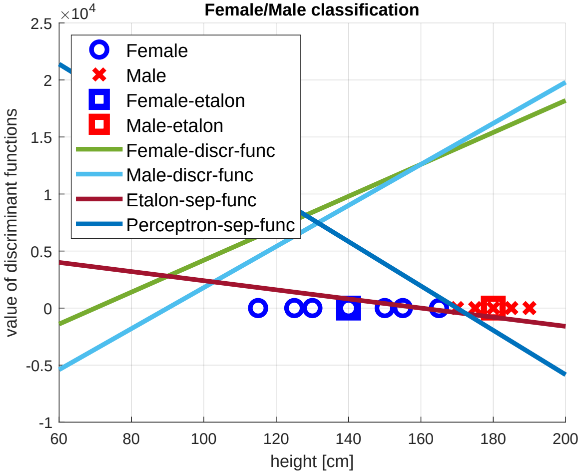
Discriminant functions for 2 classes:

$$f_F(x) = a_F x + b_F = $$
$$= e_F x - \frac{1}{2} e_F^2 = 140x - 9800$$
$$f_M(x) = a_M x + b_M = $$
$$= e_M x - \frac{1}{2} e_M^2 = 180x - 16200$$

A single discr. func. separating 2 classes:

$$g(x) = f_F(x) - f_M(x) = $$
$$= -40x + 6400$$

Example: F/M – Can we do better etalons?

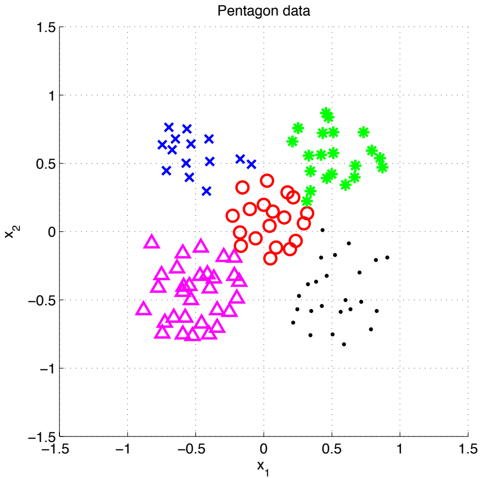


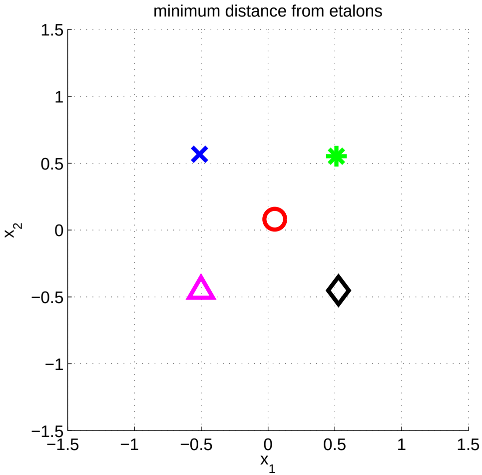
Linear classifiers based on average etalons make some errors.

A perceptron algorithm may be used to find a zero-error classifier (if one exists).

Contents

Etalons in multidimensional spaces





From $\mathcal{T} = \{(\vec{x}^{(i)}, s^{(i)})\}$, extract one etalon $\vec{e}_s$ for each class $s \in \mathcal{S}$.

Etalons in multidimensional spaces (cont.)

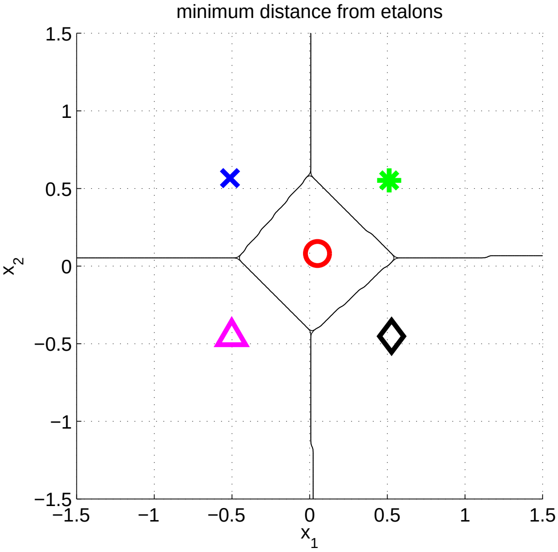
Extract etalon for each class s:

$$\vec{e}_s = \text{ave}(\{\vec{x}^{(i)} : s^{(i)} = s\})$$

Decision strategy

$$\delta(\vec{x}) = \underset{s \in S}{\text{argmin}} \|\vec{x} - \vec{e}_s\|^2$$

The corresponding decision boundaries halve the distances between pairs of etalons.



Etalons in multidimensional spaces (cont.)

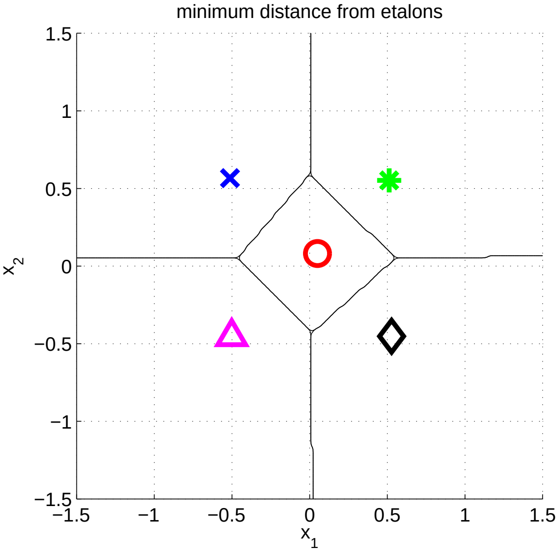
Extract etalon for each class s:

$$\vec{e}_s = \mathrm{ave}(\{\vec{x}^{(i)} : s^{(i)} = s\})$$

Decision strategy

$$\delta(\vec{x}) = \underset{s \in S}{\operatorname{argmin}} \|\vec{x} - \vec{e}_s\|^2$$

The corresponding decision boundaries halve the distances between pairs of etalons.



Etalons in multidimensional spaces (cont.)

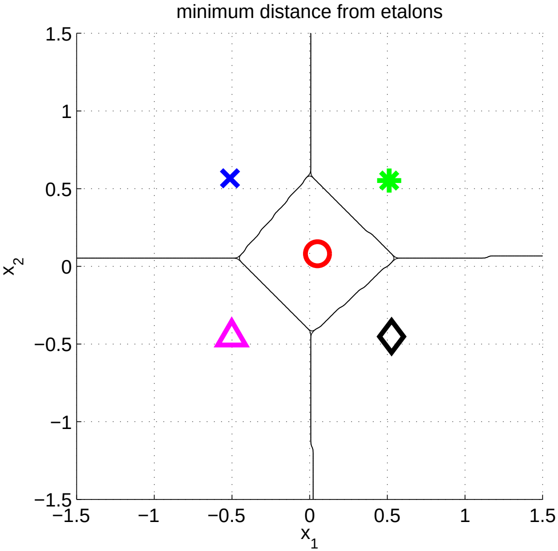
Extract etalon for each class s:

$$\vec{e}_s = \text{ave}(\{\vec{x}^{(i)} : s^{(i)} = s\})$$

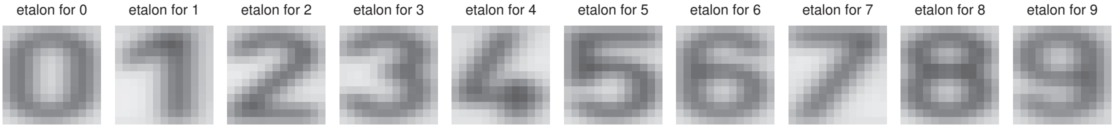
Decision strategy

$$\delta(\vec{x}) = \underset{s \in S}{\text{argmin}} \|\vec{x} - \vec{e}_s\|^2$$

The corresponding decision boundaries halve the distances between pairs of etalons.



Digit recognition – average-based etalons



Figures from [7].

Contents

Bayesian classification vs Discriminant functions

Decision based on discriminant function:

$$\delta(\vec{x}) = \underset{s \in \mathcal{S}}{\operatorname{argmax}}\, f(\vec{x}, s)$$

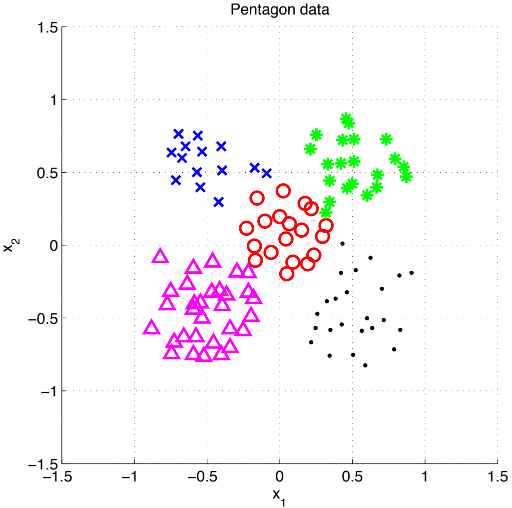
Decision based on posterior prob. (Bayes):

$$\delta(\vec{x}) = \underset{s \in \mathcal{S}}{\operatorname{argmax}}\, P(s|\vec{x}) = \underset{s \in \mathcal{S}}{\operatorname{argmax}}\, \frac{P(\vec{x} \mid s)P(s)}{P(\vec{x})}$$

If we choose

$$f(\vec{x}, s) = P(\vec{x} \mid s)P(s),$$

the two methods coincide.



Etalon classifier: generalization to higher dimensions

$$
\begin{aligned}
\delta(\vec{x}) &= \operatorname*{argmin}_{s\in S} \|\vec{x} - \vec{e}_s\|^2 = \operatorname*{argmin}_{s\in S}(\vec{x}^\top\vec{x} - 2\,\vec{e}_s^\top\vec{x} + \vec{e}_s^\top\vec{e}_s) = \\
&= \operatorname*{argmin}_{s\in S}\left(\vec{x}^\top\vec{x} - 2\left(\vec{e}_s^\top\vec{x} - \frac{1}{2}(\vec{e}_s^\top\vec{e}_s)\right)\right) = \\
&= \operatorname*{argmax}_{s\in S}\left(\vec{e}_s^\top\vec{x} - \frac{1}{2}(\vec{e}_s^\top\vec{e}_s)\right) = \\
&= \operatorname*{argmax}_{s\in S}(\vec{w}_s^\top\vec{x} + w_{s0}) = \operatorname*{argmax}_{s\in S} g_s(\vec{x}).
\end{aligned}
$$

Linear function (plus offset)

$$
g_s(\vec{x}) = \vec{w}_s^\top\vec{x} + w_{s0}, \quad \text{where} \quad \vec{w}_s = \vec{e}_s \quad \text{and} \quad w_{s0} = -\frac{1}{2}\vec{e}_s^\top\vec{e}_s.
$$

Learning and decision

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

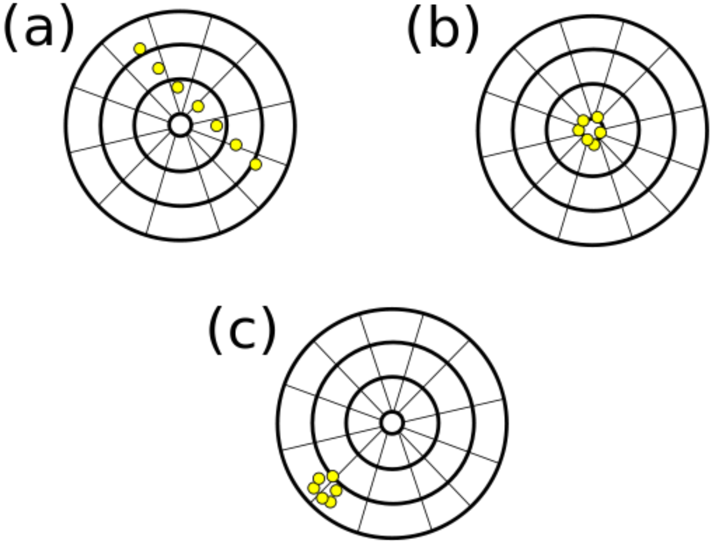
Decision stage - decide about a query $\vec{x}$.

What to learn?

- **Generative model** : Learn $P(\vec{x}, s)$. Decide according to $\mathrm{argmax}_s\, P(s|\vec{x})$.
- **Discriminative model** : Learn directly $P(s|\vec{x})$ and use it for decisions.
- **Discriminant functions** : Learn $f_s(\vec{x})$ and decide according to $\mathrm{argmax}_s\, f_s(\vec{x})$.

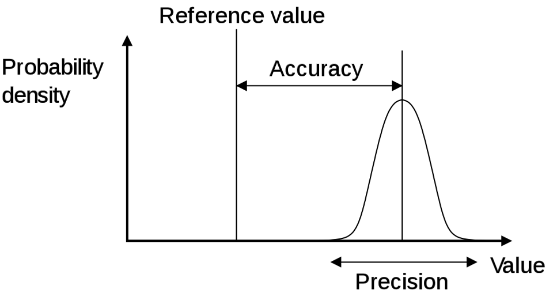
Contents

Accuracy vs precision



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

Accuracy, trueness, precision



- Trueness : closeness of the average to the correct value (systematic error, bias)
- Precision : closeness of individual measurements (variance, repeatability, reproducibility)
- Accuracy : contains both trueness and precision

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

Contents

References I

Further reading: Chapter 18 of [6], or chapter 4 of [1], or chapter 5 of [2]. Many figures created with the help of [3]. You may also play with demo functions from [7].
Human deciding and predicting under noise, [4] (in Czech [5])

[1] Christopher M. Bishop.
Pattern Recognition and Machine Learning.
Springer Science+Bussiness Media, New York, NY, 2006.
https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

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

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

References II

[4] D. Kahneman, O. Sibony, and C.R. Sunstein.
Noise: A Flaw in Human Judgment.
Little Brown Spark, 2021.

[5] D. Kahneman, O. Sibony, and C.R. Sunstein.
Šum, O chybách v lidském úsudku.
Jan Melvil Publishing, 2021.

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

[7] 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/.