WeST People and Knowledge Networks | Institute for Web Science & Technologies

UNIVERSITÄT KOBLENZ · LANDAU

Machine Learning and Data Mining WS21/22

"8 Linear Perdiction"

Dr. Zeyd Boukhers

@ZBoukhers

Institute for Web Science and Technologies

University of Koblenz-Landau

December 15, 2021

From previous lecture

- Decision Trees
- Random Forest

Agenda

- Linear regression
  - Least squares function
  - Optimization
- Linear classification
  - Perceptron classifier
  - Support Vector Machine (SVM)
  - Optimization

Linear Regression

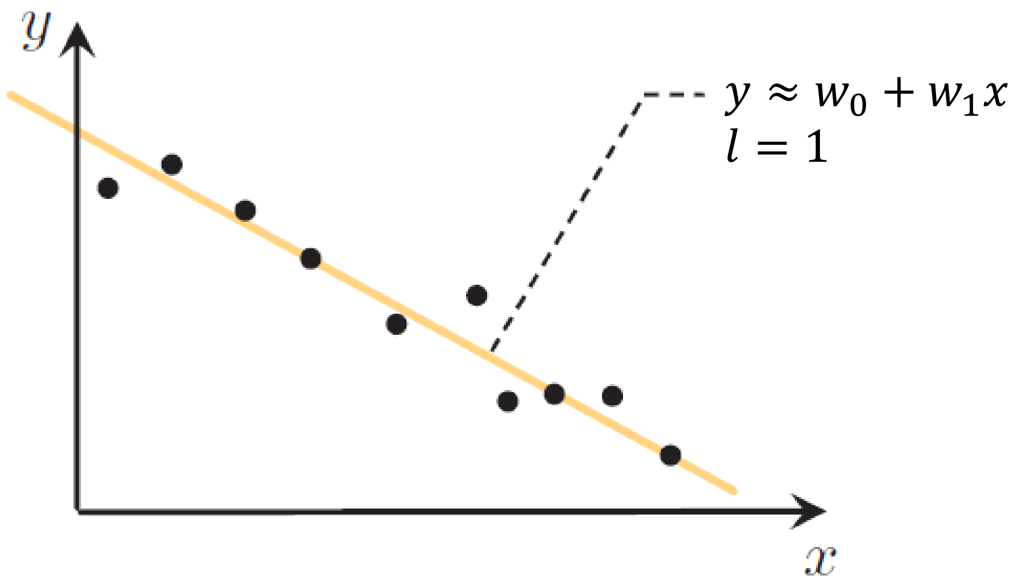
Linear regression

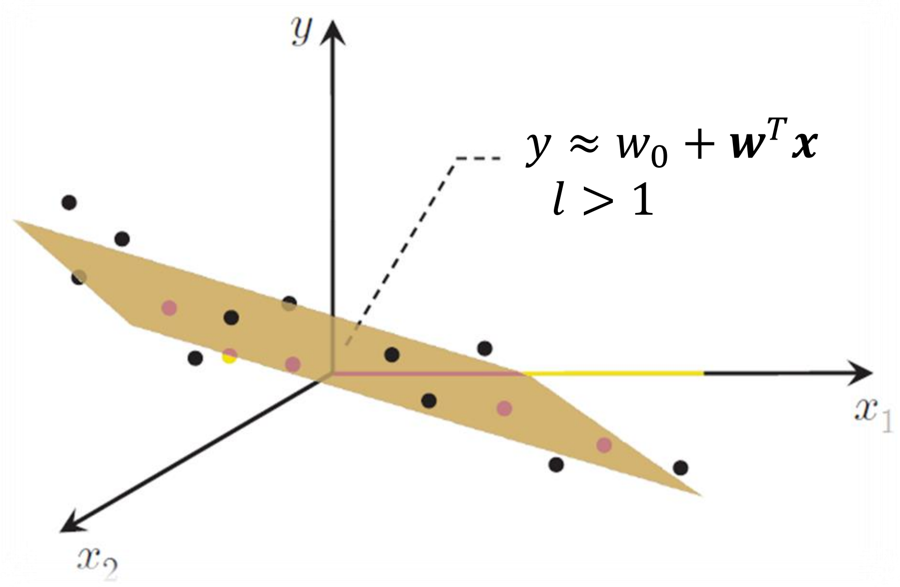
- Given N training instances: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i$ and y_i denotes the feature vector and the label (continuous measurements) of the ith instance, respectively.
- It is assumed that for any instance i, y_i is generated by an unknown rule, such as: $y_i = f(\boldsymbol{x}_i) + \varepsilon$
  - $f(.)$ is an unknown function and ε is a noise.
- The goal is to design a function $g(\boldsymbol{x})$, based on the N training instances $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$, so that the predicted value of an unseen instance $\boldsymbol{x}$ is:
  - $\hat{y} = g(\boldsymbol{x})$
  - In optimal cases, $\hat{y}$ is as close as possible to the true value y.

Linear regression

- *Linear Regression* is finding the linear function $g(\boldsymbol{x}_i) = y_i, i = 1, \dots, N$ which explains the scatter of instances best.
  - Fitting a hyperplane to the scatter of instances in $l + 1$-dimensional space.
- When $l = 1$:
  - $y \approx g(x) = w_0 + w_1 x$, where:
  - w_0 is the intercept (bias)
  - w_1 is the slope (weight).
- When $l > 1$:
  - $y \approx g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T \boldsymbol{x} = w_0 + \sum_u w_u x_u \, , u = 1, \dots, l$, where:
  - $\boldsymbol{w} = [w_1, w_2, \dots, w_l]^T$ is the vector of weights.

Linear regression

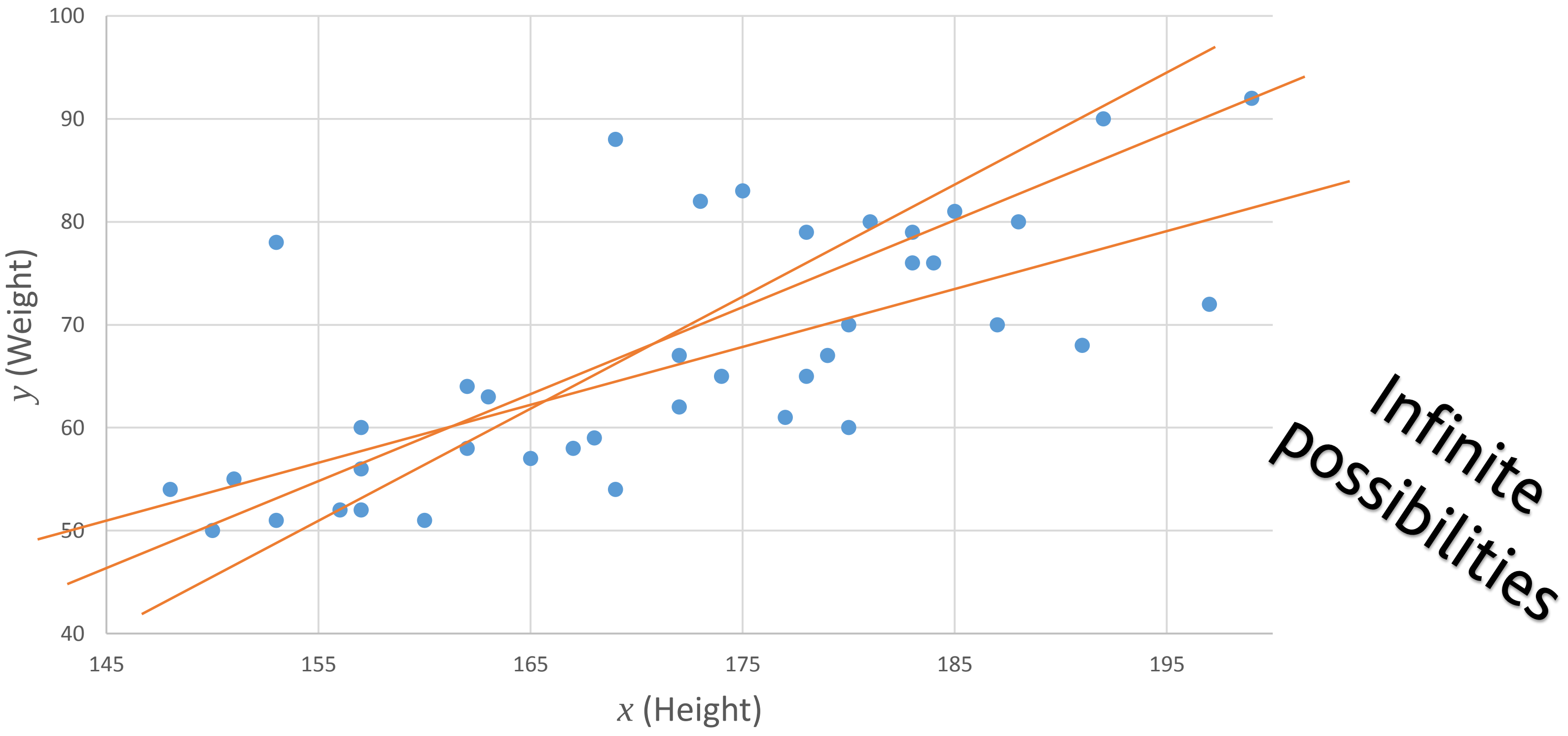




Source: Watt, J., Borhani, R., & Katsaggelos, A. K. "*Machine learning refined: foundations, algorithms, and applications*." Cambridge University Press, 2016.

Linear regression

- For an intuitive understanding, we consider $l = 1$.
- Height and Weight of 40 women (Normal body mass).

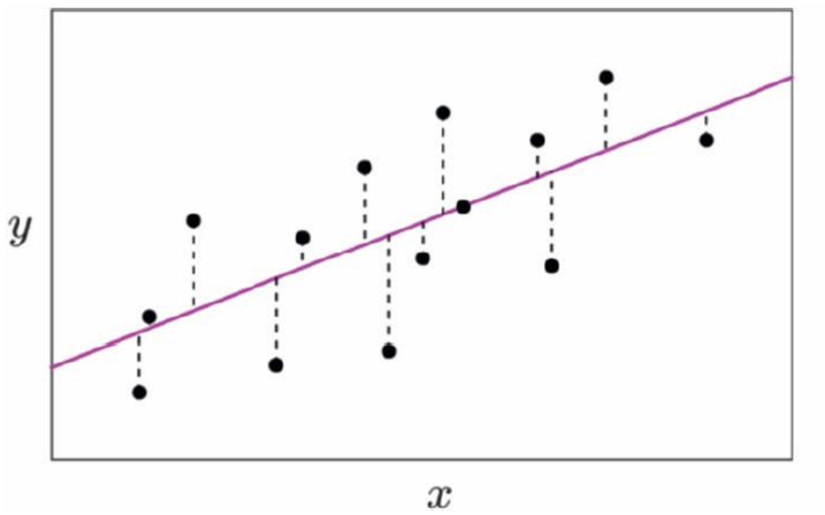


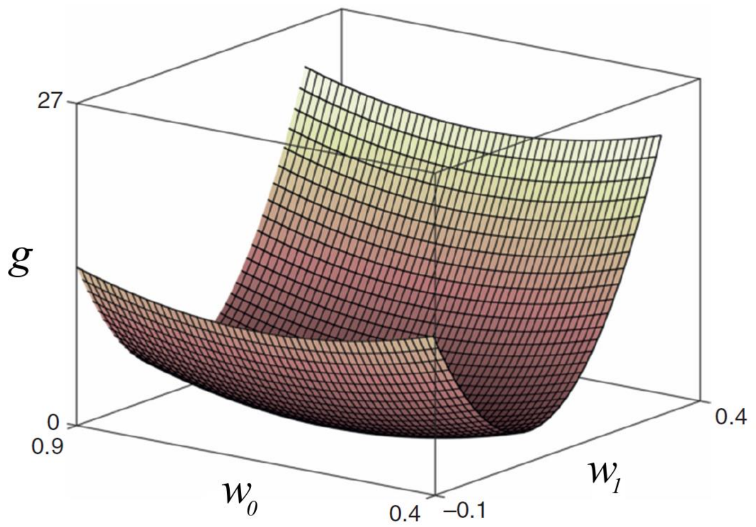
Source: https://www.kaggle.com/yersever/500-person-gender-height-weight-bodymassindex

Least Squares Cost function

- Goal: finding the parameters of the hyperplane which best fits all the training instances.
- How?
  - By computing the total squared error between the associated hyperplane and the instances.
  - The best fitting hyperplane is the one whose parameters (bias and weight) minimize this error.
- Given N instances $\{\boldsymbol{x}_i\}_{i=1}^{N}$ associated with their labels $\{y_i\}_{i=1}^{N}$ and a hyperplane $g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T\boldsymbol{x}$, the Least Squares Error is:
  - $LSE = \sum_{i=1}^{N}(y_i - g(\boldsymbol{x}_i))^2$

- Goal:
  - $\underset{w_0,\boldsymbol{w}}{\text{minimize}} \sum_{i=1}^{N} \left(y_i - (w_0 + \boldsymbol{w}^T \boldsymbol{x}_i)\right)^2$
  - *Optimization problem*
  - It has been proven that the least squares function is convex.





Source: Watt, J., Borhani, R., & Katsaggelos, A. K. “*Machine learning refined: foundations, algorithms, and applications*.” Cambridge University Press, 2016.

Optimizing the least squares cost

- Let $\widetilde{\boldsymbol{w}} = [w_0, \boldsymbol{w}]^T$ and for each instance $\boldsymbol{x}_i$, $\widetilde{\boldsymbol{x}}_i = \left[x_0, x_{i,1}, x_{i,2}, \dots, x_{i,l}\right]^T$, where x_0 is always $= 1$, LSE becomes:

$$LSE_{\widetilde{\boldsymbol{w}}} = \sum_{i=1}^{N} \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)^2$$

- By applying the chain rule:

$$\nabla LSE_{\widetilde{\boldsymbol{w}}} = \frac{\partial LSE}{\partial \widetilde{\boldsymbol{w}}} = -2 \sum_{i=1}^{N} \widetilde{\boldsymbol{x}}_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)$$

$$= -2 \sum_{i=1}^{N} \widetilde{\boldsymbol{x}}_i y_i + 2\widetilde{\boldsymbol{w}} \left(\sum_{i=1}^{N} \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T\right)$$

Remember: 1) $\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i = \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}}$

2) $LSE_{\widetilde{\boldsymbol{w}}}$ takes its minimum (not possible for the maximum) when its derivative $\nabla LSE_{\widetilde{\boldsymbol{w}}} = 0$

Optimizing the least squares cost

- From $-2\sum_{i=1}^{N}\tilde{\boldsymbol{x}}_i y_i + 2\tilde{\boldsymbol{w}}\left(\sum_{i=1}^{N}\tilde{\boldsymbol{x}}_i\tilde{\boldsymbol{x}}_i{}^T\right) = 0$, we can obtain:

$$\tilde{\boldsymbol{w}}^* = \left(\sum_{i=1}^{N}\tilde{\boldsymbol{x}}_i\tilde{\boldsymbol{x}}_i{}^T\right)^{-1}\sum_{i=1}^{N}\tilde{\boldsymbol{x}}_i y_i$$
$$= \left(\tilde{X}^T\tilde{X}\right)^{-1}\tilde{X}^T\boldsymbol{y}$$

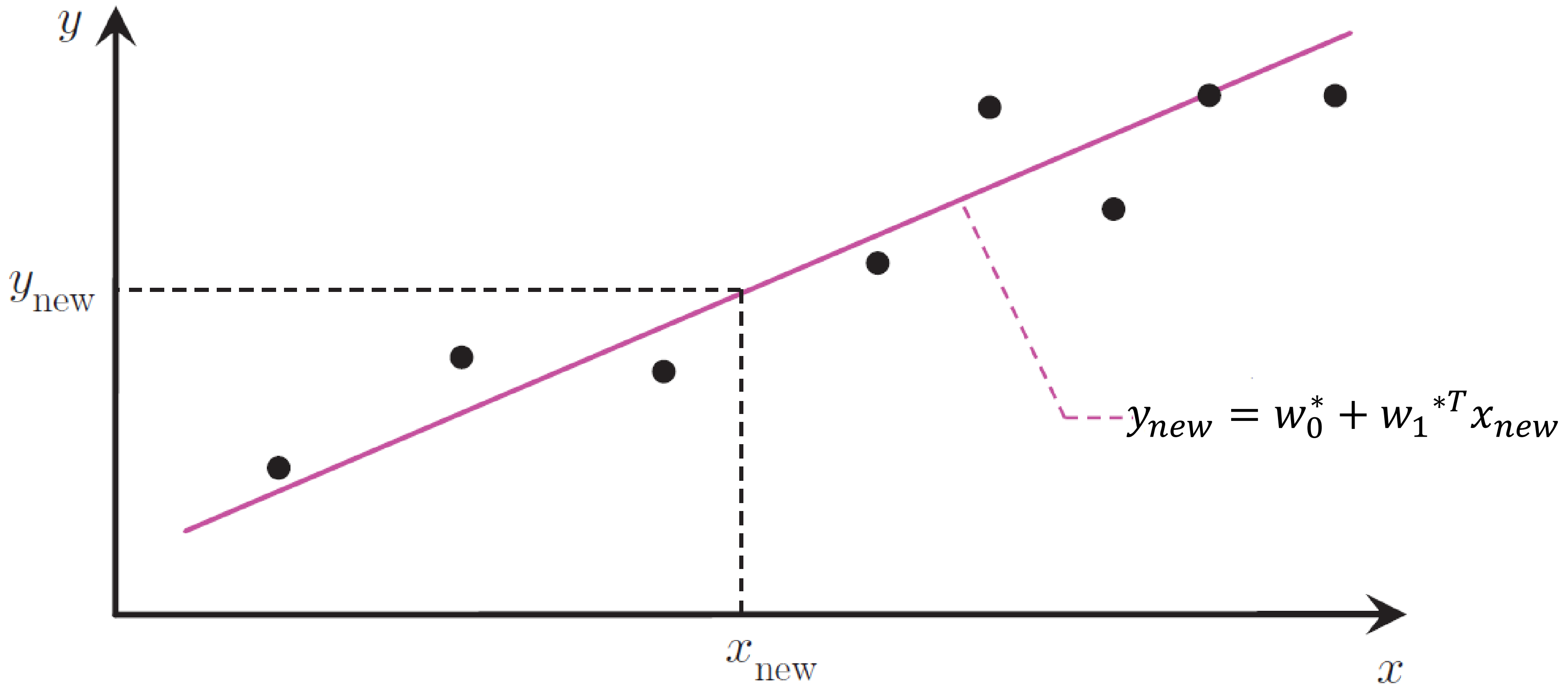
- By solving the above equation, we can obtain the optimal weight vector $\tilde{\boldsymbol{w}}^*$
- How to compute the efficacy of the linear model given $\tilde{\boldsymbol{w}}^*$?
  - We use the Mean Squared Error (MSE):
  - $MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (w_0^* + \boldsymbol{w}^{*T}\boldsymbol{x}_i)\right)^2$

Optimizing the least squares cost

- Is it always efficient to compute $\left(\tilde{X}^T\tilde{X}\right)^{-1}\tilde{X}^T\boldsymbol{y}$?
  - No, when the data is large and the number of attributes (variables) is large the computation of $\tilde{X}^T\tilde{X}$ is expensive.
  - Solution: Use gradient descent.

- How to predict the label of a new instance $\boldsymbol{x}_{new}$?

$$y_{new} = w_0^* + \boldsymbol{w}^{*T}\boldsymbol{x}_{new}$$



Source: Watt, J., Borhani, R., & Katsaggelos, A. K. “*Machine learning refined: foundations, algorithms, and applications*.” Cambridge University Press, 2016.

Example (Bonus slide)

- Let's take the example from the first lecture:
- We want to predict the price of our car based on the registration year, say 2014.
  - Obviously, all other properties (e.g. model, colour, etc.) are known.
  - We could obtain some examples of the same car from e-commerce websites:

In this example:
$l = 1$

Registration Year (x)	Price in € (y)
2010	6,000
2012	7,200
2012	8,000
2016	14,000
2017	14,500
2018	18,000
2019	20,000

Example (Bonus slide)

- For simplicity, the year will be represented by the two first digits (2010 → 10) and the price is divided by 1000 $(6000 \rightarrow 6)$.
- The goal is to find $\tilde{\boldsymbol{w}}^* = \left(\tilde{X}^T \tilde{X}\right)^{-1} \tilde{X}^T \boldsymbol{y}$

$$\tilde{X} = \begin{bmatrix} 10 & 1 \\ 12 & 1 \\ 12 & 1 \\ 16 & 1 \\ 17 & 1 \\ 18 & 1 \\ 19 & 1 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 6 \\ 7{,}2 \\ 8 \\ 14 \\ 14{,}5 \\ 18 \\ 20 \end{bmatrix}$$

Registration Year (x)	Price in € (y)
2010	6,000
2012	7,200
2012	8,000
2016	14,000
2017	14,500
2018	18,000
2019	20,000

Example (Bonus slide)

$$\tilde{X}^T\tilde{X} = \begin{bmatrix} 1618 & 104 \\ 104 & 7 \end{bmatrix}$$

$$\left(\tilde{X}^T\tilde{X}\right)^{-1} = \begin{bmatrix} 1{,}37\mathrm{E}-02 & -2{,}04\mathrm{E}-01 \\ -2{,}04\mathrm{E}-01 & 3{,}17\mathrm{E}+00 \end{bmatrix}$$

$$\tilde{X}^T\boldsymbol{y} = \begin{bmatrix} 1416{,}9 \\ 87{,}7 \end{bmatrix}$$

$$\left(\tilde{X}^T\tilde{X}\right)^{-1}\tilde{X}^T\boldsymbol{y} = \begin{bmatrix} 1{,}56E+00 \\ -1{,}07E+01 \end{bmatrix} = \tilde{\boldsymbol{w}}^*$$

Registration Year (x)	**Price in €** (y)
2010	6,000
2012	7,200
2012	8,000
2016	14,000
2017	14,500
2018	18,000
2019	20,000

Example (Bonus slide)

- Using the obtained $\widetilde{\boldsymbol{w}}^* = \begin{bmatrix} 1{,}56E + 00 \\ -1{,}07E + 01 \end{bmatrix}$
  - What is the price of the similar car registered on 2014 (2014 → 14)?
  - ➢ $\hat{y} = 14 * 1{,}56E + 00 + -1{,}07E + 01 = 11{,}18823$
  - ➢ Remember that we divided by 1000, so the estimated price is $11188{,}2$€

Registration Year (x)	Price in € (y)
2010	6,000
2012	7,200
2012	8,000
2016	14,000
2017	14,500
2018	18,000
2019	20,000

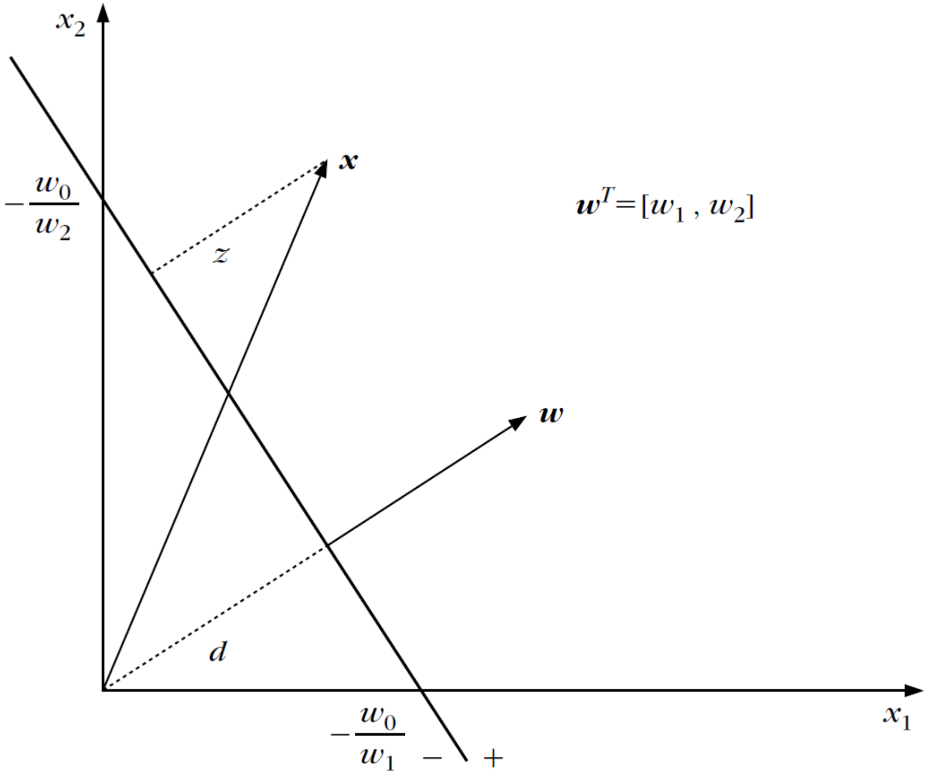
Linear Classification

Perceptron classifier

- Given N training instances: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i$ and y_i denotes the feature vector and the class of the ith instance, respectively. Each instance belongs to either ω_1 or ω_2.
- The aim is to learn a hyperplane $g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T\boldsymbol{x}$ that separates the classes; where:
  - $\boldsymbol{w} = [w_1, w_2, \dots, w_l]^T$ is the vector of weights.
  - w_0 is the threshold
- For any two instances $\boldsymbol{x}'$ and $\boldsymbol{x}''$ on the hyperplane:
  - $w_0 + \boldsymbol{w}^T\boldsymbol{x}' = w_0 + \boldsymbol{w}^T\boldsymbol{x}'' \Rightarrow \boldsymbol{w}^T(\boldsymbol{x}' - \boldsymbol{x}'') = 0$

Perceptron classifier

- From the previous equation, it is clear that the vector $\boldsymbol{w}$ is orthogonal to the decision hyperplane.

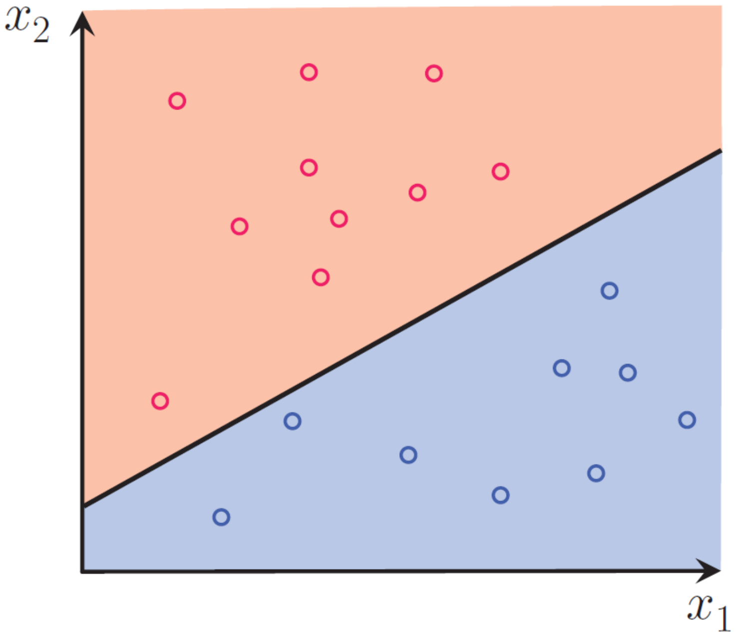


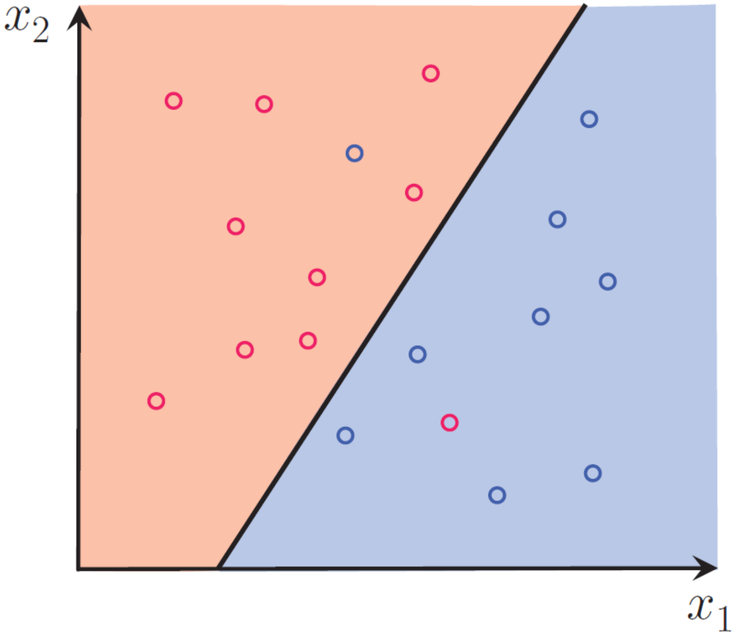
Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

Perceptron classifier

- Instead of $\boldsymbol{w} = [w_1, w_2, \dots, w_2]^T$ and $\boldsymbol{x} = [x_1, x_2, \dots, x_2]^T$, let $\widetilde{\boldsymbol{w}} = [w_0, w_1, w_2, \dots, w_2]^T$ and $\widetilde{\boldsymbol{x}} = [x_0 = 1, x_1, x_2, \dots, x_2]^T$. Then $g(\boldsymbol{x})$ becomes:
  - $g(\boldsymbol{x}) = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}$
- Goal: compute the unknown parameters $w_u, u = 0, \dots, l$ that define the hyperplane.
- Assumption: the two classes ω_1 and ω_2 are linearly separable where:
  - $\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}} > 0, \ \ \forall \boldsymbol{x} \in \omega_1,$
  - $\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}} < 0, \ \ \forall \boldsymbol{x} \in \omega_2.$

Perceptron classifier

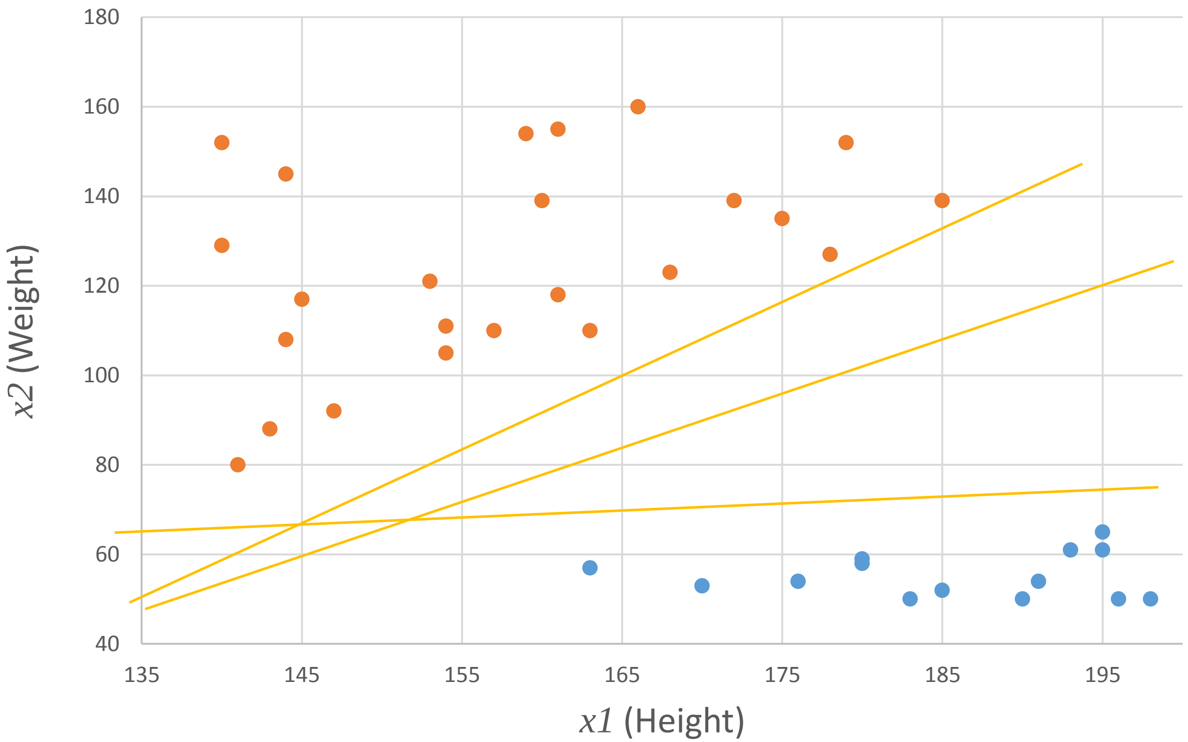




Source: Watt, J., Borhani, R., & Katsaggelos, A. K. "*Machine learning refined: foundations, algorithms, and applications*." Cambridge University Press, 2016.

Perceptron classifier

- For an intuitive understanding, we consider $l = 1$.
- Height and Weight of 14 women and 24 men.



Which line?

Perceptron cost function

- **Goal**: finding the parameters of the hyperplane which splits all the training instances w.r.t their classes.
  - Optimization task
- The perceptron cost:

$$J(\tilde{\boldsymbol{w}}) = \sum_{\boldsymbol{x} \in Y} \left(\delta_{\boldsymbol{x}} \tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}} \right)$$

- Where:
  - Y is the set of misclassified instances, given $\tilde{\boldsymbol{w}}$
  - $\delta_{\boldsymbol{x}} = \begin{cases} -1 & \text{if } \boldsymbol{x} \in \omega_1 \\ +1 & \text{if } \boldsymbol{x} \in \omega_2 \end{cases}$
- We can notice that:
  - $J(\tilde{\boldsymbol{w}})$ is always positive
  - $J(\tilde{\boldsymbol{w}}) = 0 \text{ iff } Y = \emptyset$

Perceptron cost function

- The cost $J(\tilde{\boldsymbol{w}})$ is continuous and piecewise linear.
  - By smoothly changing $\tilde{\boldsymbol{w}}$, $J(\tilde{\boldsymbol{w}})$ changes linearly until the number of instacens in Y changes.
  - The gradient function is discontinuous
  - ➢An iterative minimization of $J(\tilde{\boldsymbol{w}})$ is adopted (*gradient descent*):

$$\tilde{\boldsymbol{w}}(t+1) = \tilde{\boldsymbol{w}}(t) - \rho t \left.\frac{\partial J(\tilde{\boldsymbol{w}})}{\partial \tilde{\boldsymbol{w}}}\right|_{\tilde{\boldsymbol{w}}=\tilde{\boldsymbol{w}}(t)}$$

  - Where
    - $\tilde{\boldsymbol{w}}(t)$ is the weight estimate at the tth iteration step.
    - ρt is a sequence of positive real numbers (learning rates)
    - It is not valid at the points of discontinuity.
    - $\frac{\partial J(\tilde{\boldsymbol{w}})}{\partial \tilde{\boldsymbol{w}}} = \sum_{x \in Y} \delta_x \boldsymbol{x}$

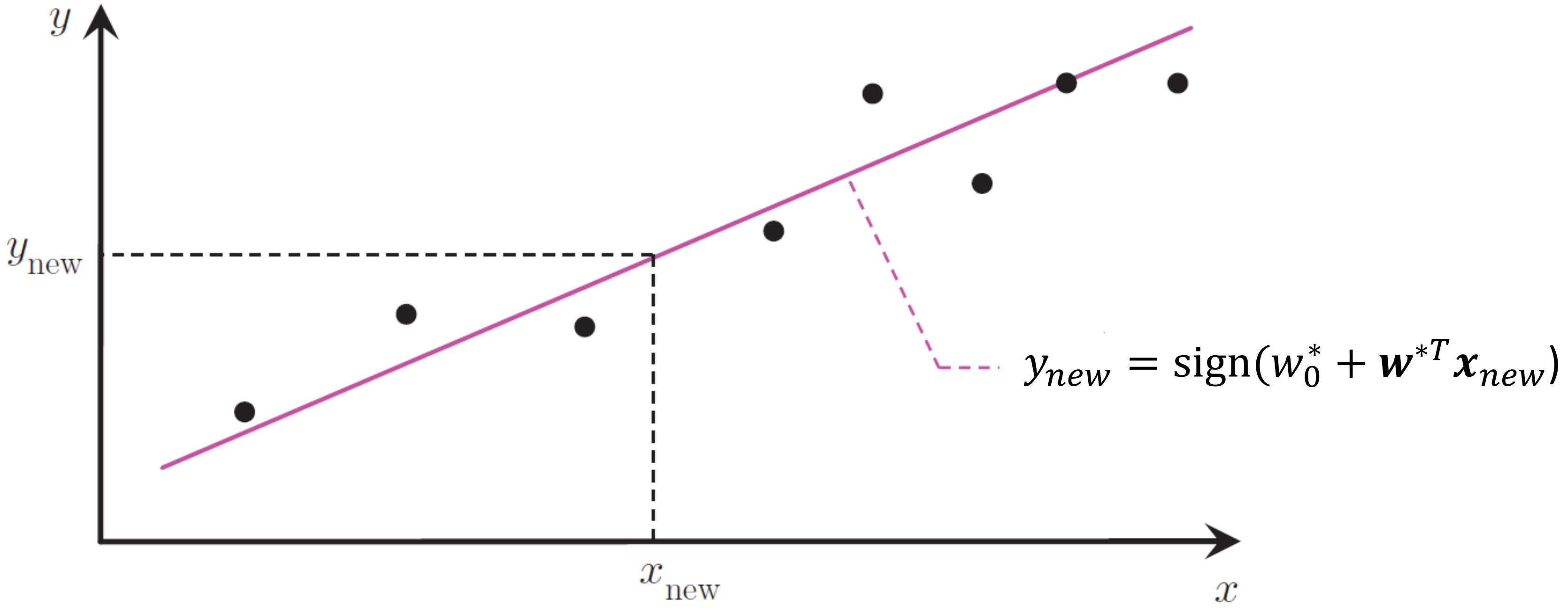
Perceptron classifier: Pseudo code

- $t = 0$
- Choose $\tilde{\boldsymbol{w}}(0)$ randomly
- Choose $\rho 0$
- Repeat:
  - $Y = \emptyset$
  - For $i = 1$ to N
    - If $\left(\delta_{\boldsymbol{x}_i} \tilde{\boldsymbol{w}}(t)^T \boldsymbol{x}_i\right) \geq 0$ then $Y = Y \cup \{\boldsymbol{x}_i\}$
  - $\tilde{\boldsymbol{w}}(t+1) = \tilde{\boldsymbol{w}}(t) - \rho t \sum_{\boldsymbol{x} \in Y} \delta_{\boldsymbol{x}} \boldsymbol{x}$
  - Adjust ρt
  - $t \mathrel{+}= 1$
- Until $Y = \emptyset$ (when the data is perfectly linearly separable)

- How to predict the class of a new instance $\boldsymbol{x}_{new}$?

$$\widetilde{\boldsymbol{w}}^* = \widetilde{\boldsymbol{w}}(\text{last})$$

$$\gamma(\boldsymbol{x}_{new}) = \text{sign}\left(w_0^* + \boldsymbol{w}^{*T}\boldsymbol{x}_{new}\right)$$

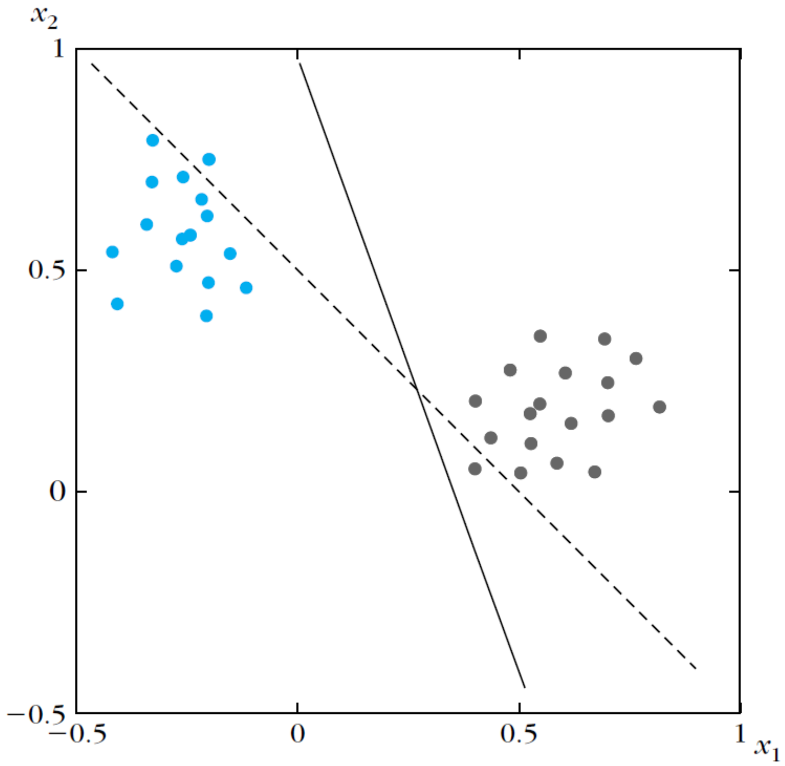


Source: Watt, J., Borhani, R., & Katsaggelos, A. K. "*Machine learning refined: foundations, algorithms, and applications*." Cambridge University Press, 2016.

Example

- Consider that at an iteration t, $\widetilde{\boldsymbol{w}}(t) = [-0.5,1,1\ ]^T$ and $\rho t = 0.7$
- $Y \neq \emptyset$
- $Y = \begin{Bmatrix} [0.4, \ \ -0.05]^T \\ [-0.2, \ \ 0.75]^T \end{Bmatrix}$
- $\widetilde{\boldsymbol{w}}(t+1) = \widetilde{\boldsymbol{w}}(t) - \rho t \sum_{\boldsymbol{x} \in Y} \delta_{\boldsymbol{x}} \boldsymbol{x}$

$$\widetilde{\boldsymbol{w}}(t+1)$$
$$= [-0.51,1\ ]^T - 0.7(-1)[1,0.4,-0.05]^T$$
$$- 0.7(+1)[1,-0.2,0.75]^T$$
$$= [-0.5,1.42,0.51\ ]^T$$



Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

Support Vector Machine (SVM)

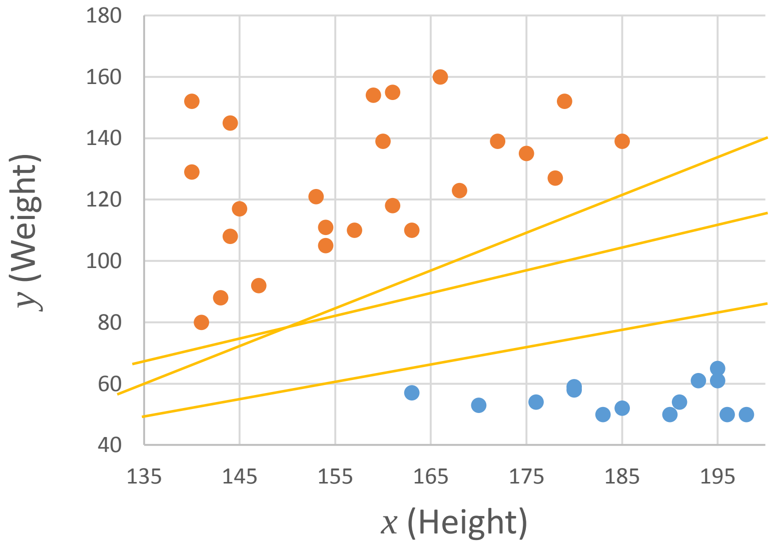
SVM

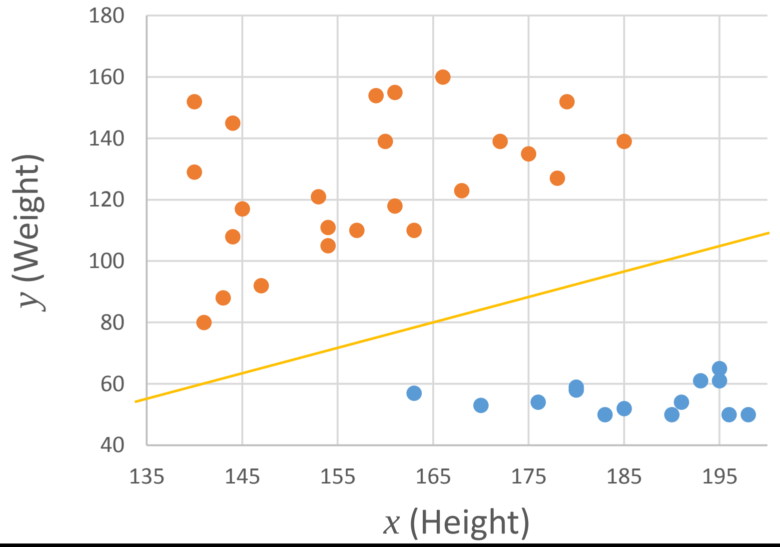
- Given N training instances: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i$ and y_i denotes the feature vector and the class of the ith instance, respectively. Each instance belongs to either ω_1 or ω_2.
- The aim is to learn a hyperplane $g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T\boldsymbol{x} = 0$ that separates *best* the classes; where:
  - $\boldsymbol{w} = [w_1, w_2, \dots, w_l]^T$ is the vector of weights.
  - w_0 is the threshold
- For any two instances $\boldsymbol{x}'$ and $\boldsymbol{x}''$ on the hyperplane:
  - $w_0 + \boldsymbol{w}^T\boldsymbol{x}' = w_0 + \boldsymbol{w}^T\boldsymbol{x}'' = \boldsymbol{w}^T(\boldsymbol{x}' - \boldsymbol{x}'') = 0$

Isn't it the same as the perceptron classifier?

Perceptron vs. SVM

- Perceptron:
  - The obtained solution is not unique. The algorithm may converge to any solution.
  - It can run online and update when new instances show up.
- SVM
  - Only one solution can be obtained.
  - It runs only after collecting all the training instances

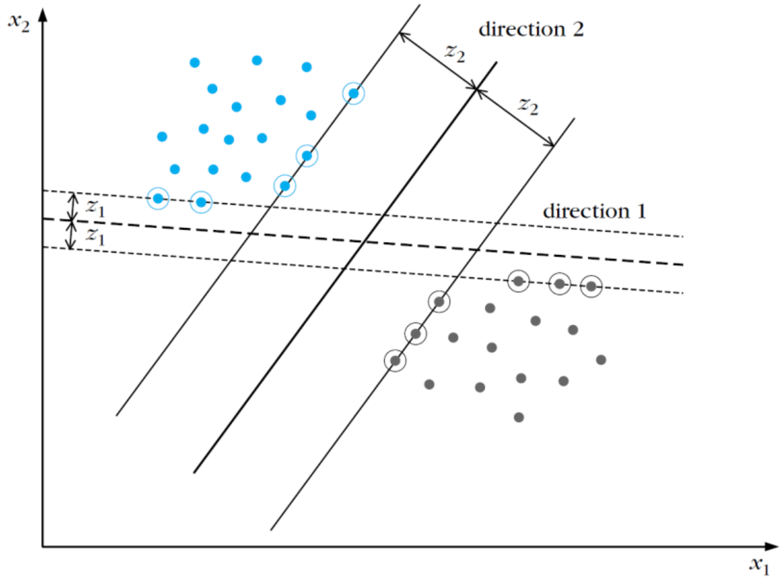




SVM

- Assuming that there is one optimal solution, SVM finds the “best” of all possible separating hyperplanes.
  - It maximizes the margins.
  - the distance between the evenly spaced instances that just touch each class.

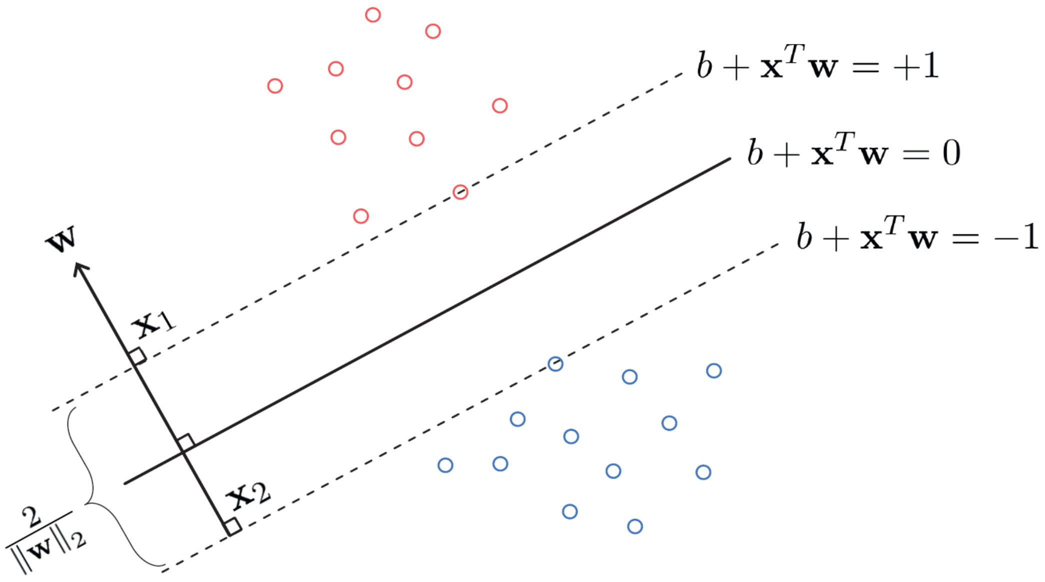
- In other words, the best hyperplane is the one that leaves the maximum margin from both classes.



Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

- **Goal**: search for the direction that gives the maximum possible margin.
- For an hyperplane $w_0 + \boldsymbol{w}^T\boldsymbol{x} = 0$, the width of the buffer zone confined between two symmetric margins is written as $w_0 + \boldsymbol{w}^T\boldsymbol{x} = \pm 1$ (each margin just touching one of the two classes)

- The sum of margins $= 2z$
- Remember:
  - $d(\boldsymbol{x}', g(\boldsymbol{x})) = \frac{|g(\boldsymbol{x}')|}{||\boldsymbol{w}||}$



Source: Watt, J., Borhani, R., & Katsaggelos, A. K. "*Machine learning refined: foundations, algorithms, and applications*." Cambridge University Press, 2016.

SVM

- Let's scale $\boldsymbol{w}$ and w_0 so that $g(\boldsymbol{x})$ at the nearest instance in one class $= 1$ and thus $= -1$ for the instance in the other class.
  - $\frac{1}{||\boldsymbol{w}||} + \frac{1}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||}$
- With the condition that:
  - $\boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 1, \ \forall \boldsymbol{x} \in \omega_1$
  - $\boldsymbol{w}^T\boldsymbol{x} + w_0 \leq -1, \ \forall \boldsymbol{x} \in \omega_2$
- Let y_i be a class indicator $(+1 \text{ for } \omega_1, -1 \text{ for } \omega_2)$ for the instance $\boldsymbol{x}_i$

SVM

- The task becomes to compute $\boldsymbol{w}, w_0$ of the hyperplane so that:
  - minimize $J(\boldsymbol{w}, w_0) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2$
  - Subject to $y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0\right) \geq 1,\ \ i = 1,2,\dots,N$
    - Inequality constraints.

➢A quadratic optimization task subject to a set of linear inequality constraints.

➢Use Karush Kuhn Tucker (KKT).

Karush Kuhn Tucker

- How to optimize $J(\boldsymbol{w}, w_0)$ with its constraints $\text{cons}(\boldsymbol{w}, w_0) = a$?
- We consider $\mathcal{L}\,(\boldsymbol{w}, w_0, \boldsymbol{\lambda})$

$$\mathcal{L}\,(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = J(\boldsymbol{w}, w_0) - (\boldsymbol{\lambda}(\text{cons}(\boldsymbol{w}, w_0) - a)$$
$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{i=1}^{N} \lambda_i\left[y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0\right) - 1\right]$$

- Where λ_i is the Lagrange Multiplier.
- The minimizer of $J(\boldsymbol{w}, w_0)$ has to satisfy:
  - $\frac{\partial}{\partial \boldsymbol{w}}\mathcal{L}\,(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = 0$
  - $\frac{\partial}{\partial w_0}\mathcal{L}\,(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = 0$

Finding the optimum

- These equations result in:

$$\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i$$

- Where
  - $\sum_{i=1}^{N} \lambda_i y_i = 0$
  - $\boldsymbol{x}_i$ when $\lambda_i \neq 0, i = 1,2, \ldots, N$ are called support vectors.
  - λ_i is positive if $\boldsymbol{x}_i$ is a support vector, 0 otherwise.
- w_0 can be then implicitly obtained from:
  - $\lambda_i \left[y_i \left(\boldsymbol{w}^T \boldsymbol{x}_i + w_0 \right) - 1 \right] = 0$

Lagrangian duality

- We need now to compute the involved parameters $\boldsymbol{\lambda}$.

$$\text{maximize } \mathcal{L}\left(\boldsymbol{w}, w_0, \boldsymbol{\lambda}\right)$$

- Subject to:
  - $\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i$
  - $\sum_{i=1}^{N} \lambda_i y_i = 0$
  - $\boldsymbol{\lambda} \geq 0$
- Using the obtained equations and a bit of algebra:

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right)$$

- Subject to:
  - $\sum_{i=1}^{N} \lambda_i y_i = 0$
  - $\boldsymbol{\lambda} \geq 0$

Prediction

- How to predict the class of a new instance $\boldsymbol{x}_{new}$?

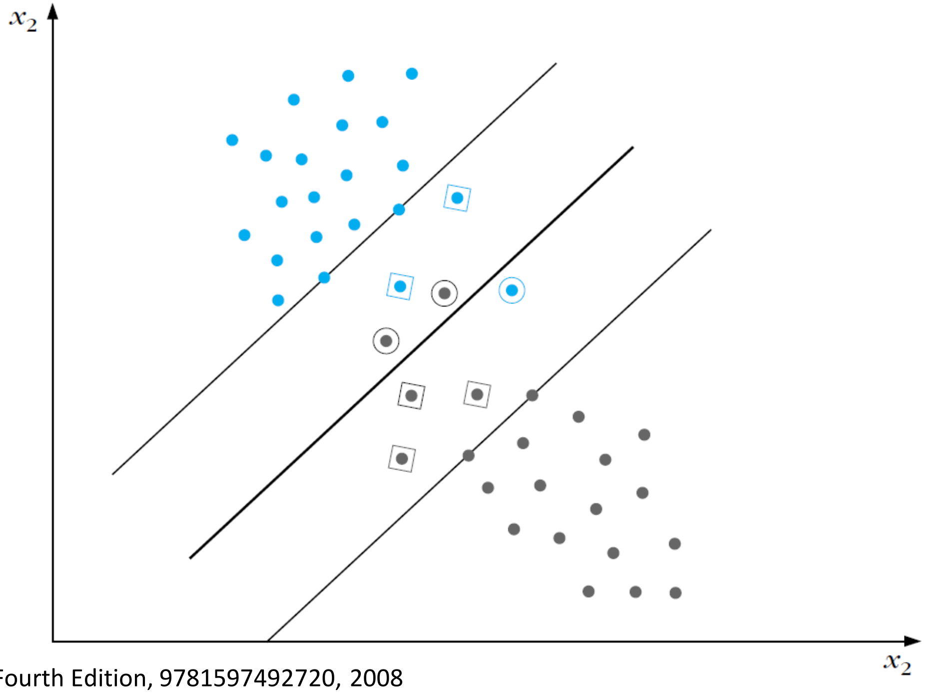
$$\gamma(\boldsymbol{x}_{new}) = \mathrm{sign}(g(\boldsymbol{x}_{new})) = \mathrm{sign}(w_0 + \boldsymbol{w}^T \boldsymbol{x}_{new})$$

$$= \mathrm{sign}\left(w_0 + \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_{new} \right)$$

Non-separable Classes

Non-separable case

- When the classes are not separable, the setup that we saw is no longer valid.
  - No hyperplane can satisfy the constraints.



Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

Non-separable case

- Considering the optimal solution, we observe:
  - Instances that comply with the constraints $\Rightarrow$ They fall where they are supposed to fall.
  - Instances that do not comply with the constraints but they still fall on the correct side.
    - $0 \leq y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) < 1$

$\xi_i = 0$

  - Instances that fall on the wrong side.
    - $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) < 0$

$0 < \xi_i \leq 1$

- All these cases can be treated by introducing a new set of variables.
  - $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) \geq 1 - \xi_i$

$\xi_i > 1$

ξ_i called slack variable

Non-separable case

- Goal: make the margin as large as possible but at the same time to keep the number of instances with $\xi_i > 0$ as small as possible.

$$J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

- Where:
  - $I(\xi_i) = \begin{cases} 1 & \text{if } \xi_i > 0 \\ 0 & \text{if } \xi_i = 0 \end{cases}$
  - C is a positive constant that controls the relative influence of the two competing terms.

Non-separable case

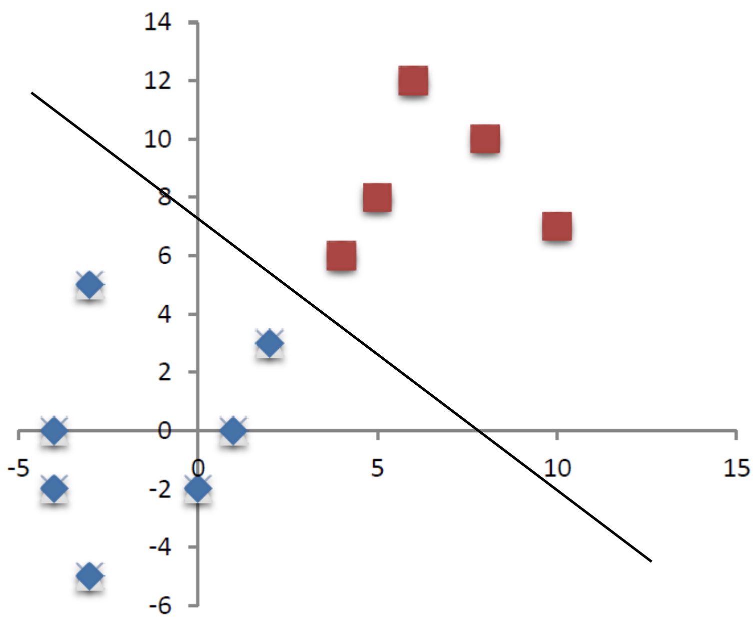
- The task becomes now:
  - minimize $J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$
  - Subject to
    - $y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0\right) \geq 1 - \xi_i,\ \ i = 1,2,\dots,N$
    - $\xi_i \geq 0,\ \ i = 1,2,\dots,N$
- the corresponding Lagrangian becomes:

$$\mathcal{L}\left(\boldsymbol{w}, w_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}\right) = J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) - \boldsymbol{\lambda}(\mathrm{cons1}(\boldsymbol{w}, w_0) - a) - \boldsymbol{\mu}(\mathrm{cons2} - \beta)$$

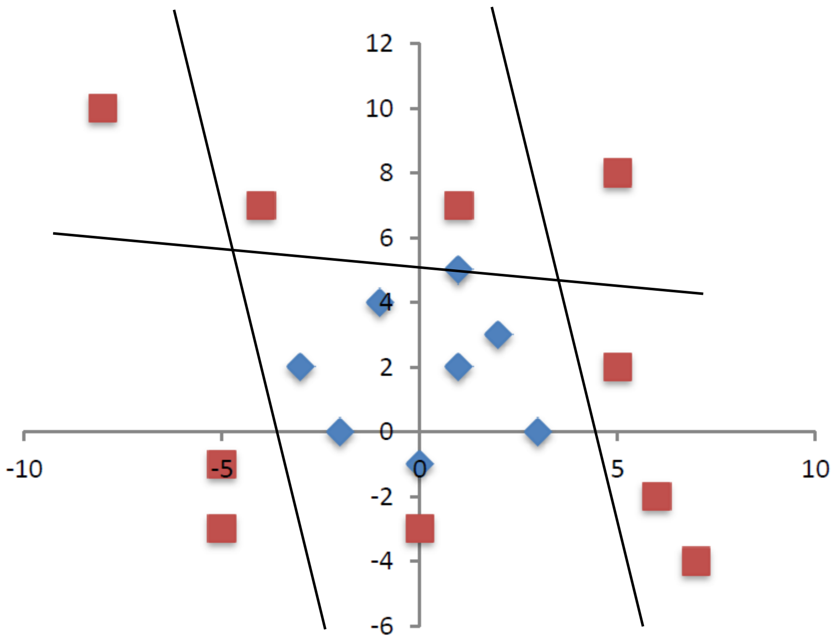
$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N} I(\xi_i) - \sum_{i=1}^{N}\lambda_i\left[y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0\right) - 1 + \xi_i\right] - \sum_{i=1}^{N}\mu_i\xi_i$$

Non-linearly separable Classes

Non-linearly separable classes

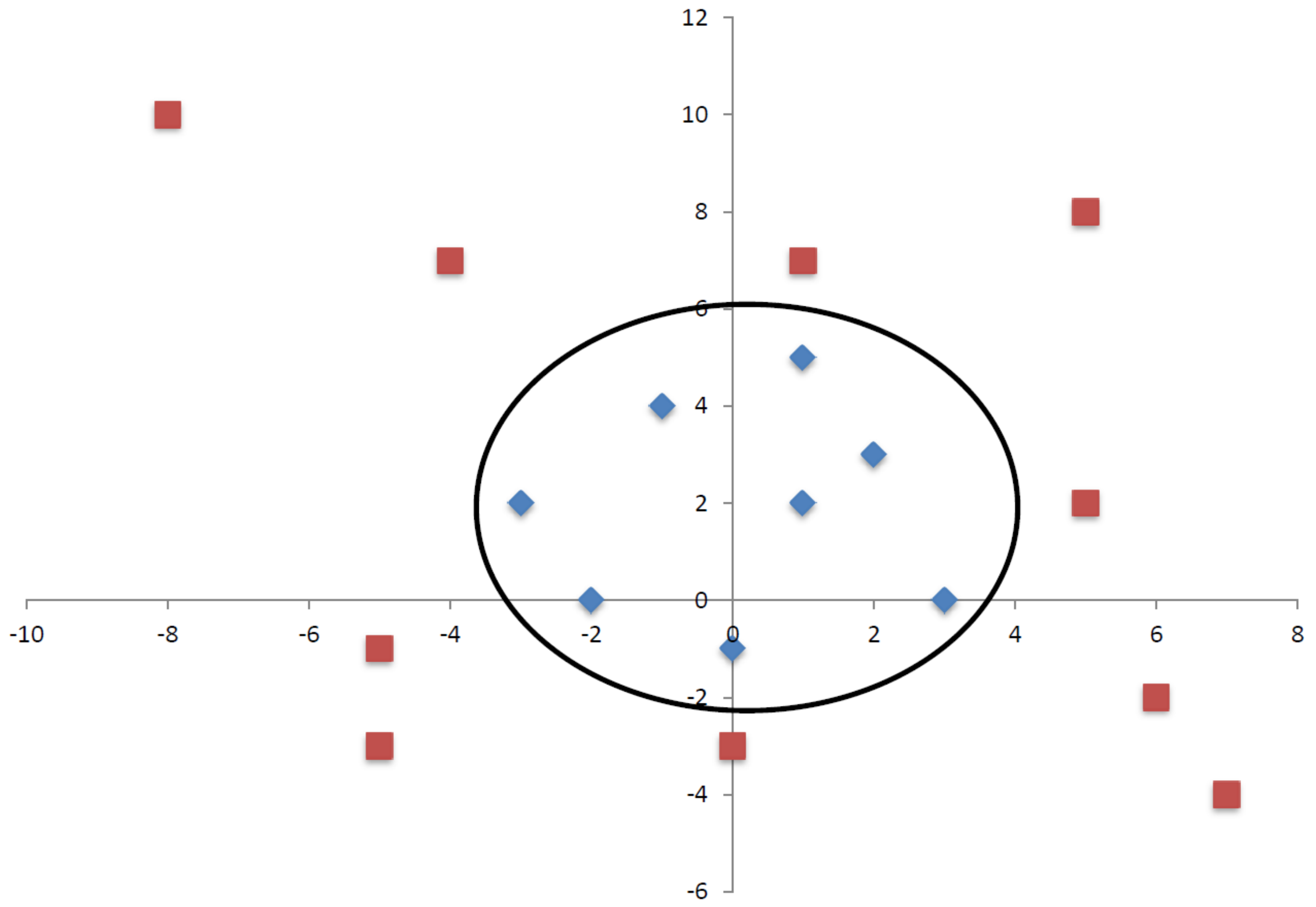


Linearly Separable

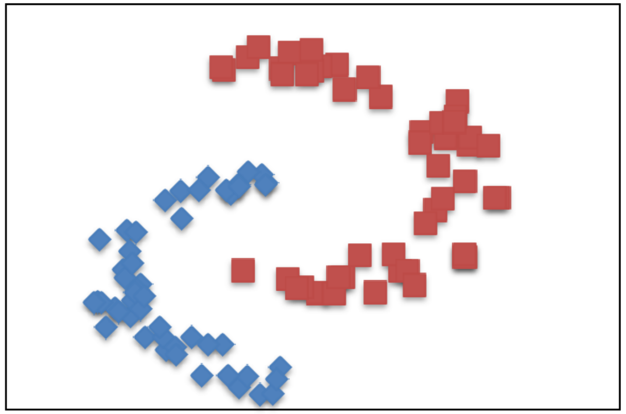


Non-Linearly Separable

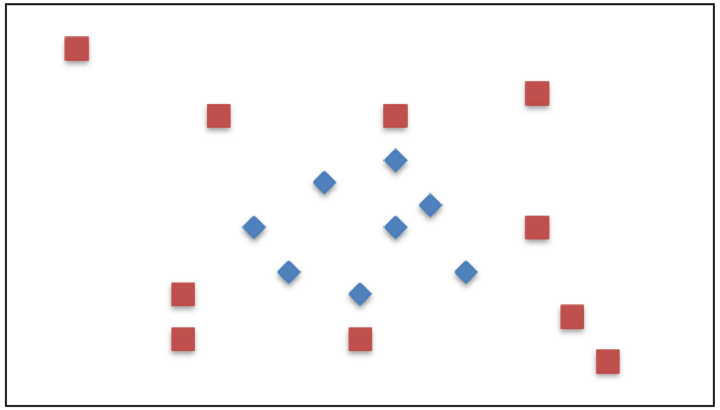
Non-linearly separable classes



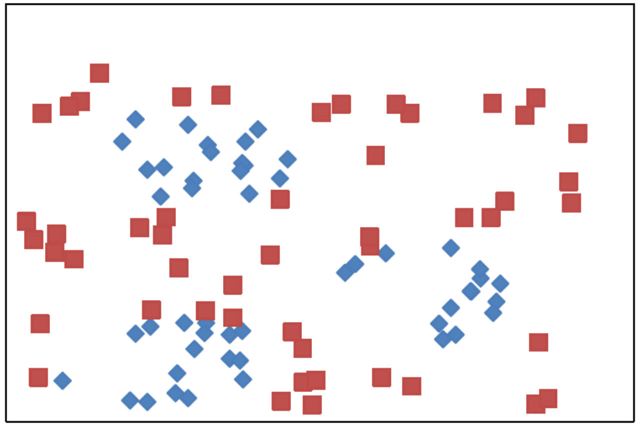
Examples of non-linear problems



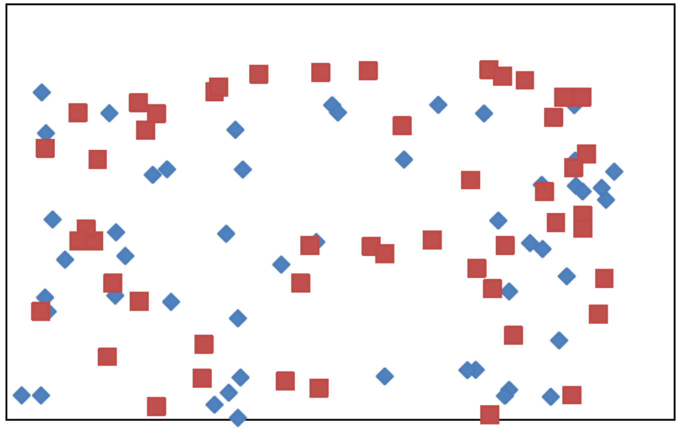
Interference



Inclusion



Multiple-Inclusion

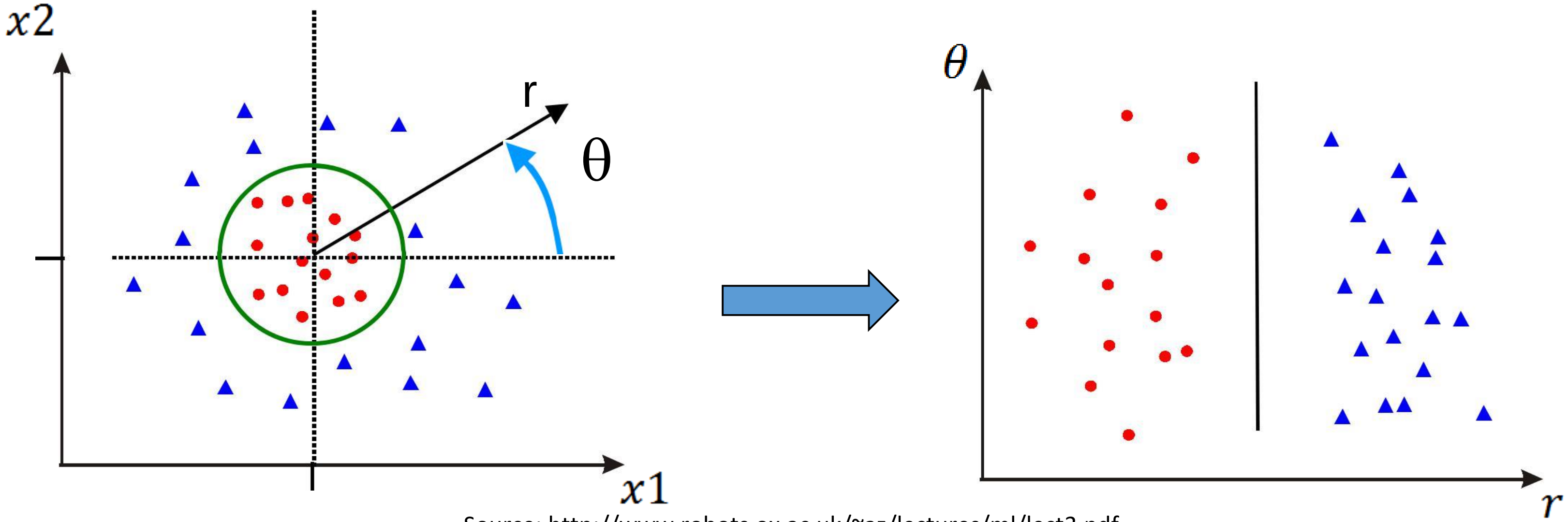


Indistinctive

Solution1: Polar coordinates

- Data is linearly separable in polar coordinates.

$$\Phi: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} r \\ \theta \end{pmatrix} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

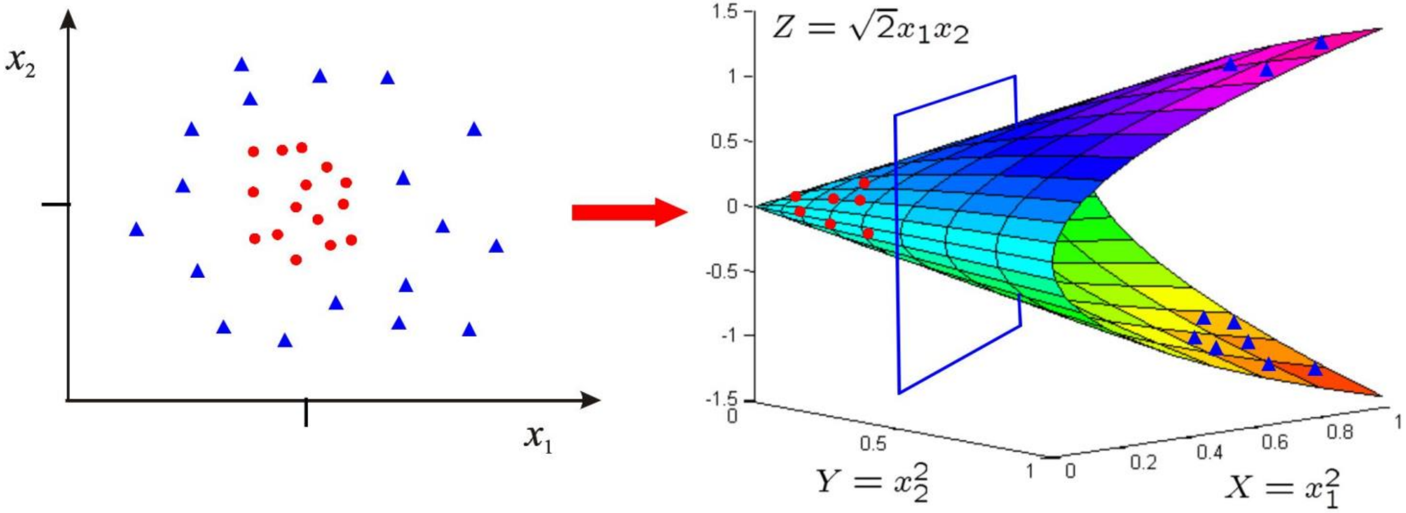


Source: http://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf

Solution2: Map data into a higher dimensional space

- Data is linearly separable in 3D
  - The problem can still be solved by a linear classifier.
  - ➢ Map the data into a higher dimensional space.
  - ➢ $\Phi: \boldsymbol{x} \rightarrow \Phi(\boldsymbol{x}): \mathbb{R}^d \rightarrow \mathbb{R}^D$

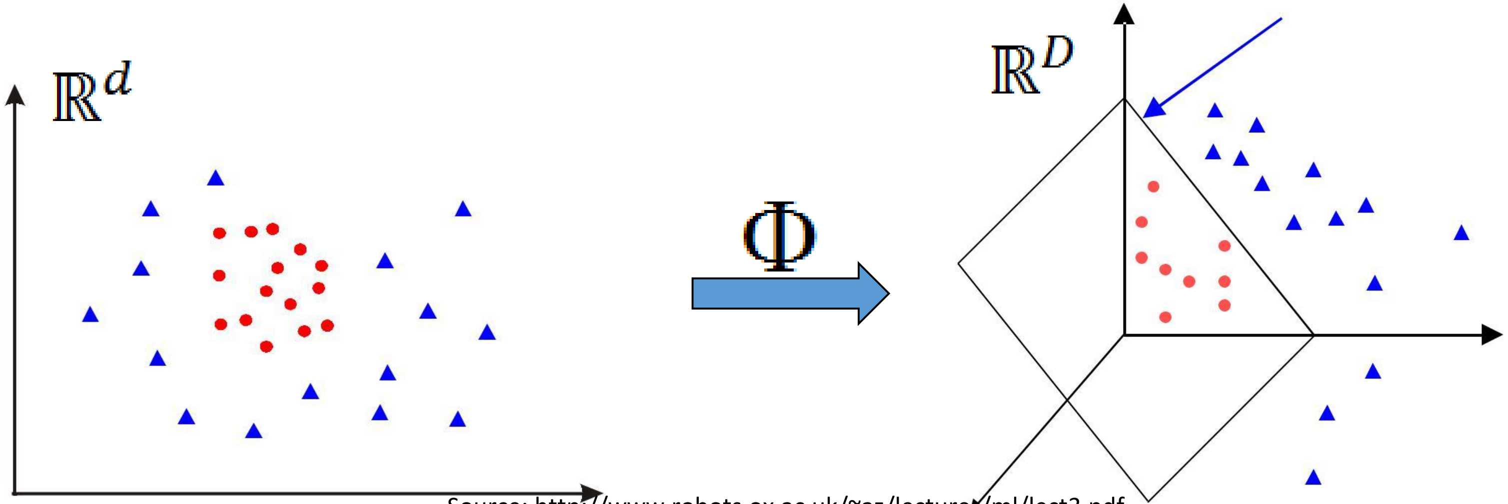
E.g. $\Phi: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \boldsymbol{x} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}: \mathbb{R}^2 \rightarrow \mathbb{R}^3$



Solution2: Map data into a higher dimensional space

- Remember:
  - $\gamma(\boldsymbol{x}_{new}) = \text{sign}\left(w_0 + \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_{new}\right)$
- The linear classifier in a higher dimensional space
  - $\gamma(\boldsymbol{x}_{new}) = \text{sign}\left(w_0 + \sum_{i=1}^{N} \lambda_i y_i \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_{new})\right)$

Similar learning process



Source: http://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf

Kernel trick

- Let $\Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_{new}) = K(\boldsymbol{x}_i, \boldsymbol{x}_{new})$
  - $K(.,.)$ is known as Kernel Function
- The classifier becomes:
  - $\gamma(\boldsymbol{x}_{new}) = \text{sign}\left(w_0 + \sum_{i=1}^{N} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_{new})\ \right)$
  - $K(.,.)$ is directly incorporated in the learning and classification function.

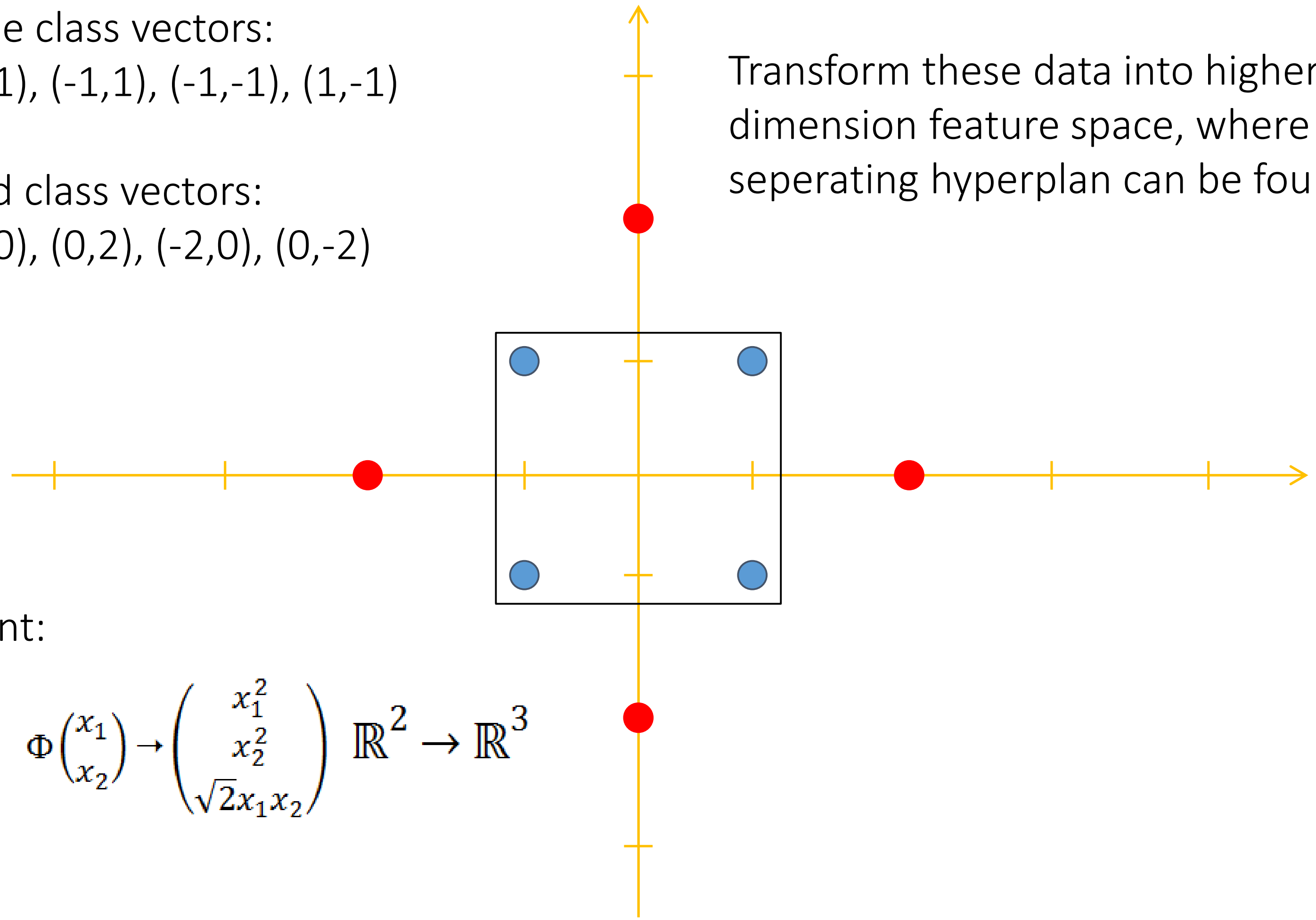
Example kernels

- Linear kernels
  - $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$
- Polynomial kernels
  - $K(\boldsymbol{x}, \boldsymbol{y}) = \left(1 + \boldsymbol{x}^T \boldsymbol{y}\right)^d$ for any $d > 0$
- Gaussian kernels
  - $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}\right)$ for $\sigma > 0$

Bonus slide

Blue class vectors:
(1,1), (-1,1), (-1,-1), (1,-1)

Red class vectors:
(2,0), (0,2), (-2,0), (0,-2)



Transform these data into higher dimension feature space, where a seperating hyperplan can be found

Hint:

$$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

Would be interesting to self study

- Multiclass case (SVM)
  - One vs One
  - One vs All

Summary

Summary

- Linear Regression
- Perceptron Classifier
- Support Vector Machine
  - Non-separable classes
  - Non-linearly separable Classes
  - Kernel trick

Thank you!

Zeyd Boukhers

E-mail: Boukhers@uni-koblenz.de
Phone: +49 (0) 261 287-2765
Web: Zeyd.Boukhers.com

University of Koblenz-Landau
Universitätsstr. 1
56070 Koblenz