Linear Classifiers

Szymon Bobek

Institute of Applied Computer science
AGH University of Science and Technology

http://geist.agh.edu.pl

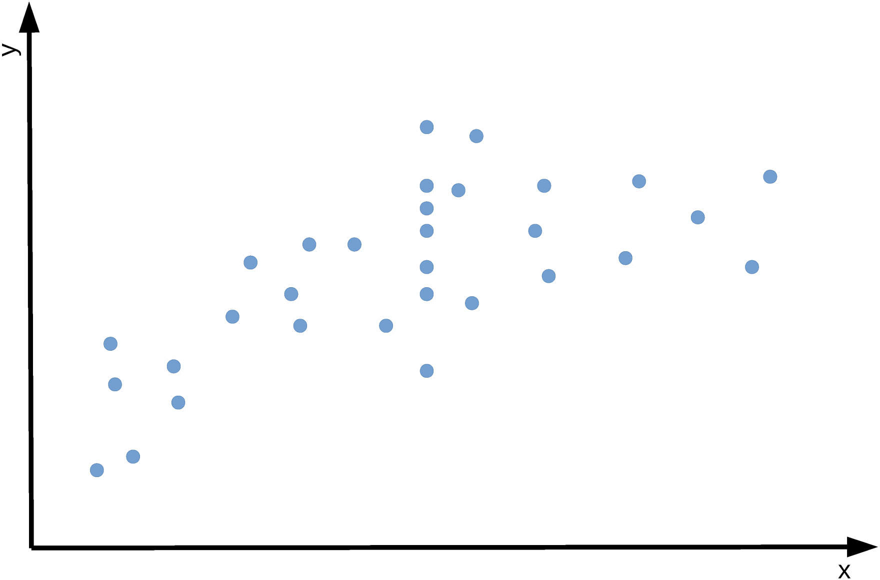
Outline I

1. Regression for classification
2. Logistic regression
   - Intuition for logistic regression
   - Cost function
   - Multi-class classification
   - Categorical values
   - Precision/Recall/ROC
3. Support Vector Machine
   - Basic linear algebra
   - Intuition behind SVM
   - Finding the margin
   - Optimizing cost function
4. Kernels
   - Intuition for kernels
   - Dual representation
   - Kernels

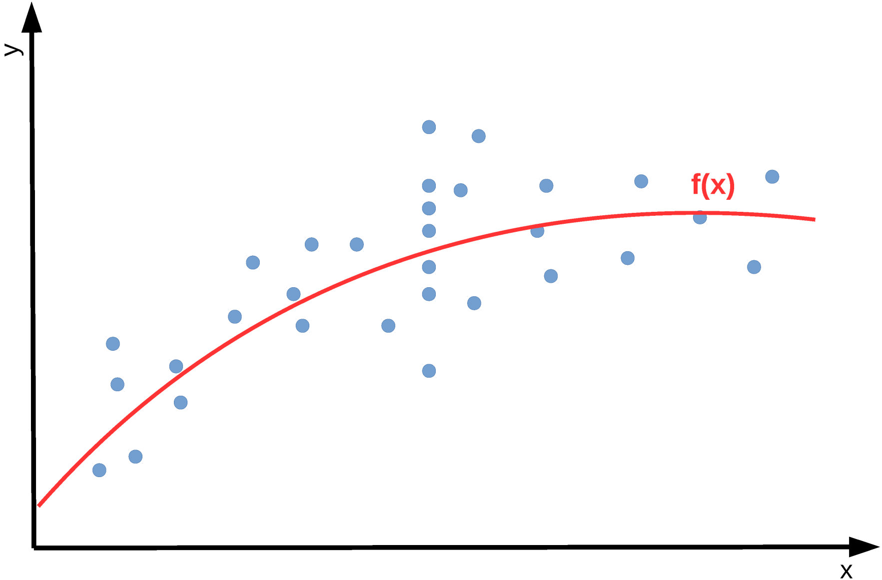
Presentation Outline

1. Regression for classification
2. Logistic regression
3. Support Vector Machine
4. Kernels

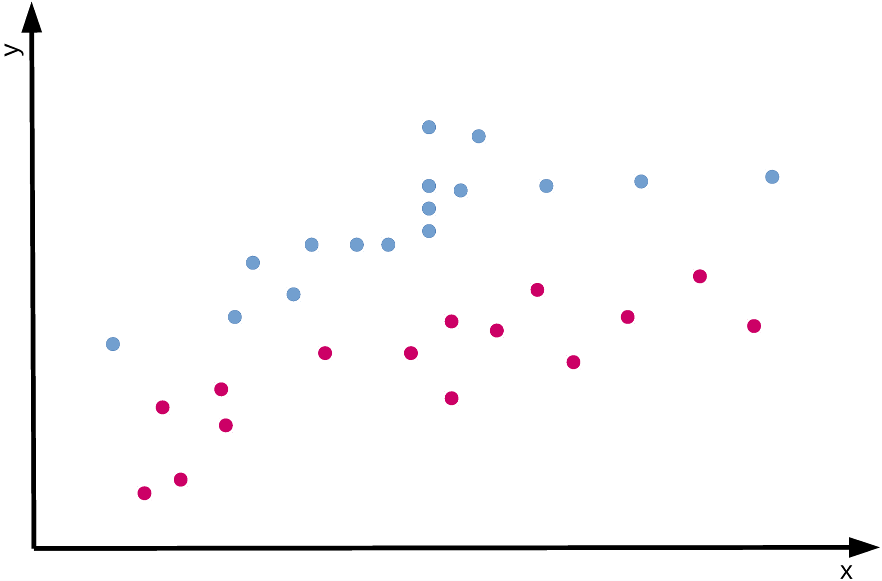
How fitting a line can be used for classification



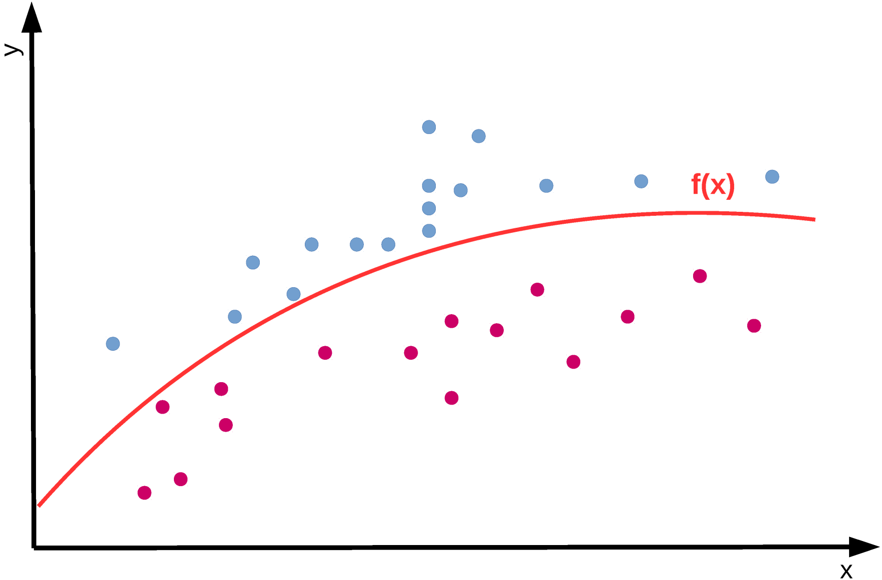
How fitting a line can be used for classification



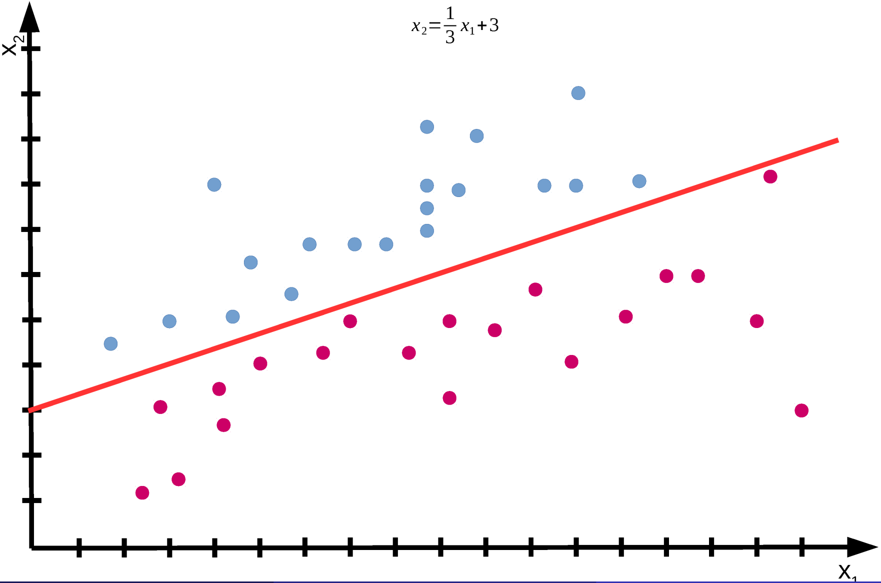
How fitting a line can be used for classification



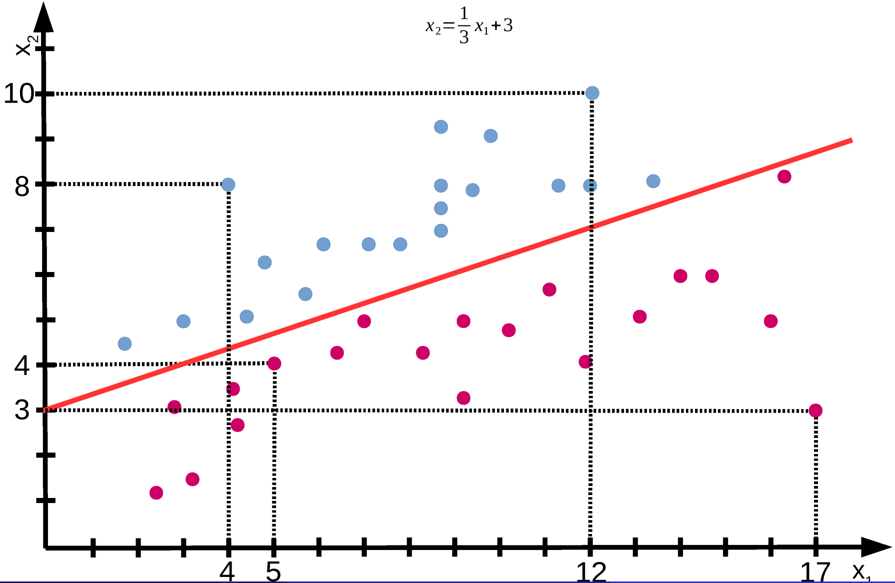
How fitting a line can be used for classification



How fitting a line can be used for classification

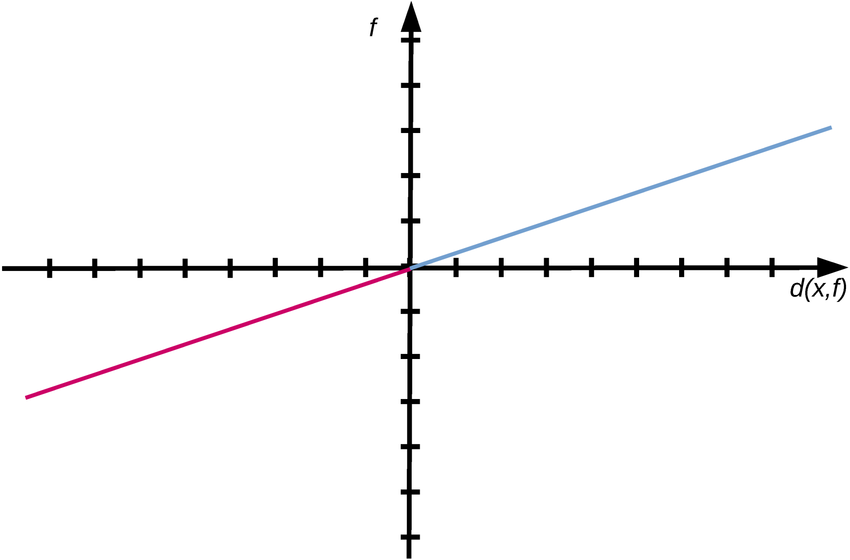


How fitting a line can be used for classification

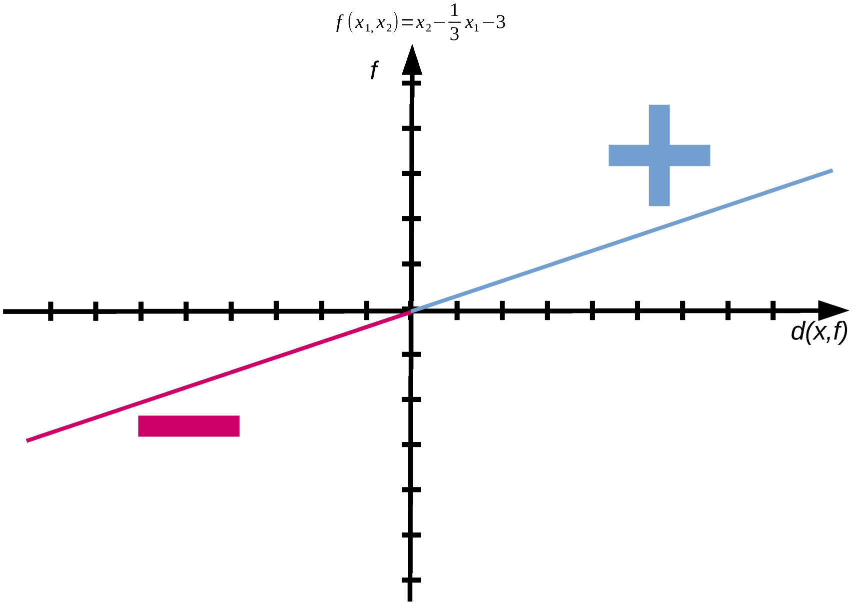


Decision boundary

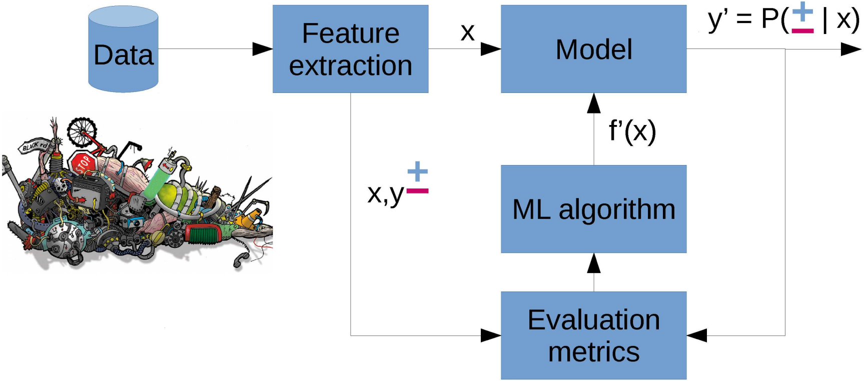
$$f(x_1, x_2) = x_2 - \frac{1}{3}x_1 - 3$$



Decision boundary



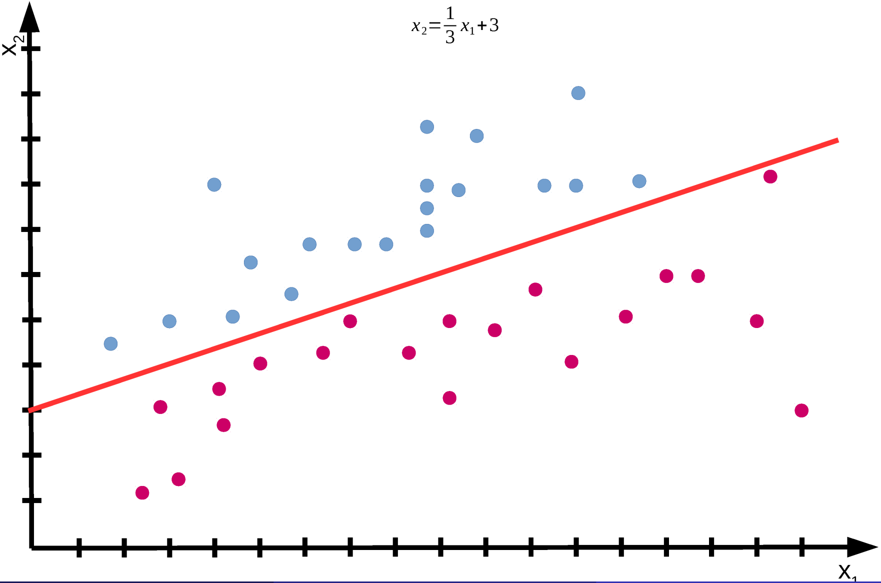
Input/output



Presentation Outline

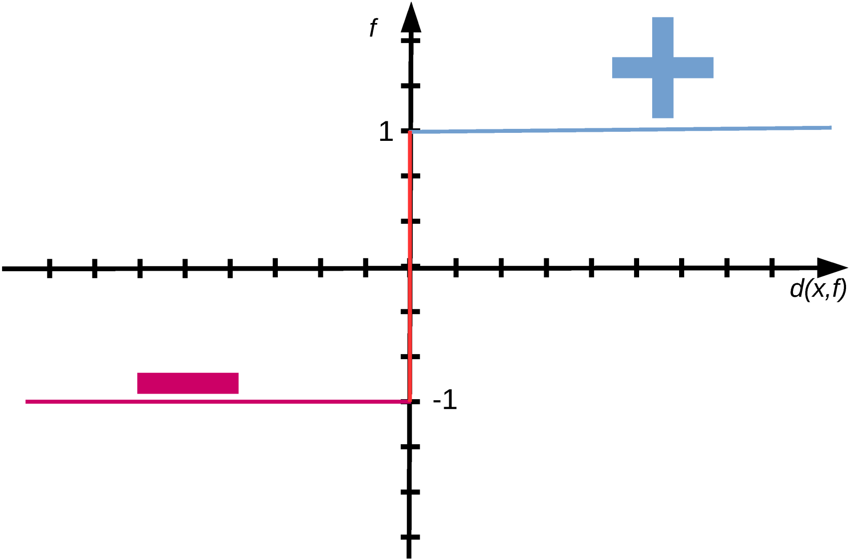
1. Regression for classification
2. Logistic regression
   - Intuition for logistic regression
   - Cost function
   - Multi-class classification
   - Categorical values
   - Precision/Recall/ROC
3. Support Vector Machine
4. Kernels

Losing information when using sing only

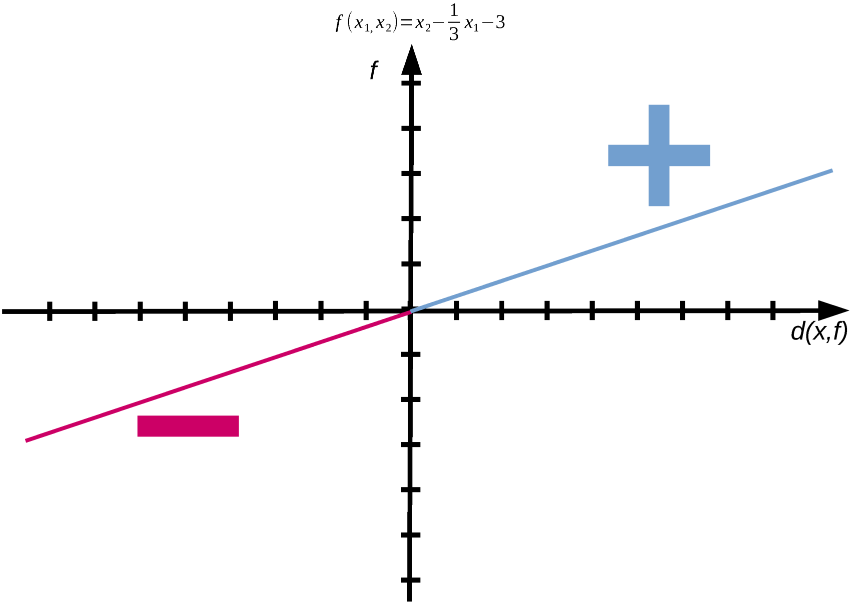


Losing information when using sing only

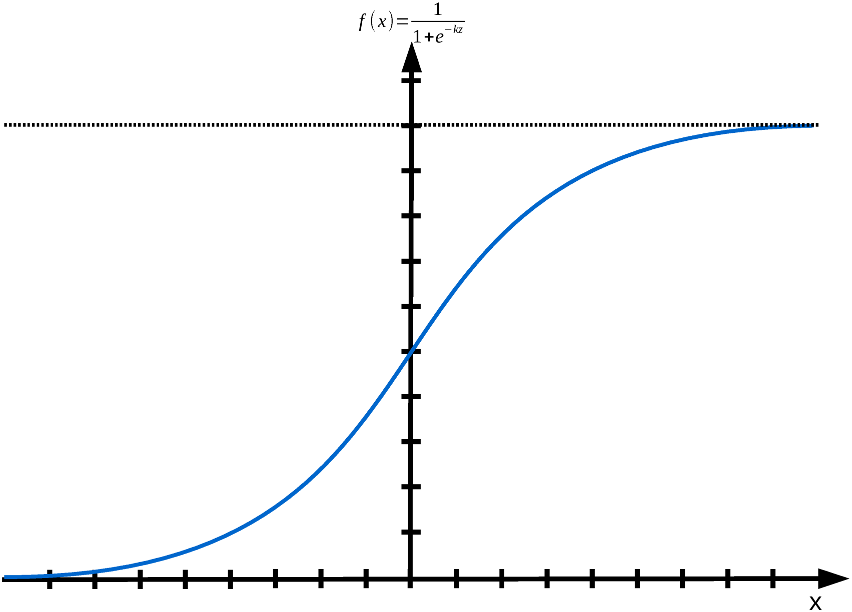
$$f(x_1, x_2) = x_2 - \frac{1}{3}x_1 - 3$$



Losing information when using sing only

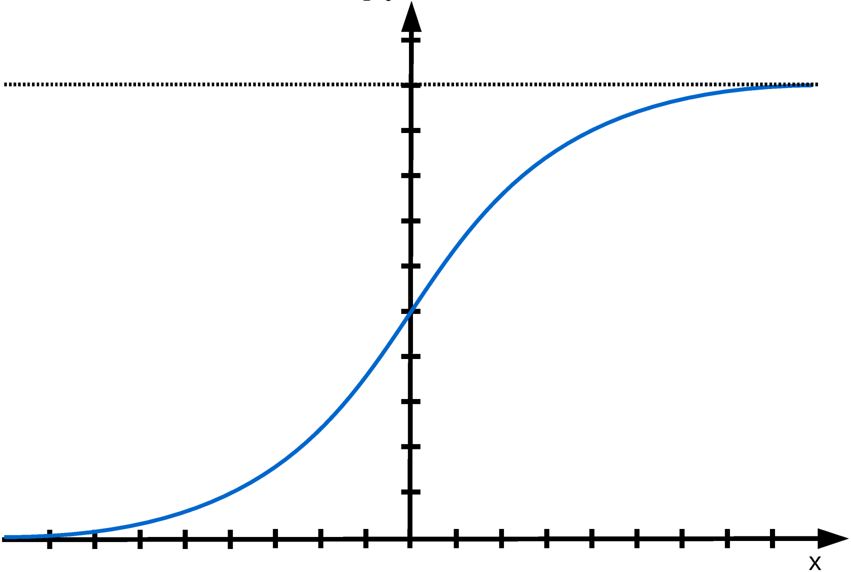


Logistic function



Logistic function

$$f(x) = \frac{1}{1+e^{-\theta^T x}} = P(y=1|\theta, x)$$



Outline

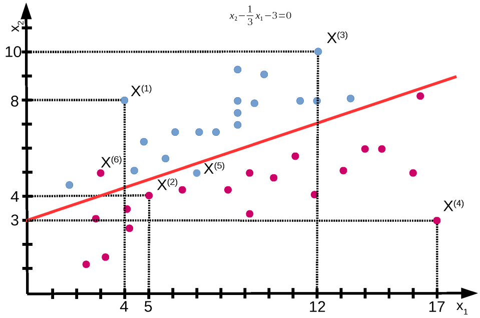
Cost function

- We train linear regression equation embedded into logistic function
- We do not have numbers as an output, but classes instead.
- Can we still use MSE for loss calculation?
- Can we use gradient/coordinate descent algorithms (is the cost function convex)?
- How to calculate the gradient?
- What is our optimization objective?

Probabilistic perspective

Optimization objective

- Sigmoid function returns $P(y = 1|\theta x)$
- Therefore $P(y = -1|\theta x) = 1 - P(y = 1|\theta x)$
- We want to select such θ, so that the probability that given training example belongs to its true class is highest:



x_1	x_2	y	Max
4	8	1	
5	4	-1	
12	10	1	
17	3	-1	
7	5	1	
3	5	-1	

Maximize (log)likelihood

- We maximize $P(y = 1/-1|x, \theta)$ for every datapoint, so we have:

$$\underbrace{\max_{\theta} \prod_{i=1}^{N} P(y^{(i)}|x^{(i)}, \theta)}_{\ell(\theta)}$$

- Machine learning loves logarithms, so instead we have:

$$\max_{\theta} ln \prod_{i=1}^{N} P(y^{(i)}|x^{(i)}, \theta) = \underbrace{\max_{\theta} \sum_{i=1}^{N} lnP(y^{(i)}|x^{(i)}, \theta)}_{\ell\ell(\theta)}$$

- And finally:

$$\max_{\theta} \ell\ell(\theta) = \max_{\theta} \sum_{i=1}^{N} \Big[\mathbb{1}[y = +1] lnP(y^{(i)} = +1|x^{(i)}, \theta) + \\ + \mathbb{1}[y = -1] lnP(y^{(i)} = -1|x^{(i)}, \theta) \Big]$$

Simplifying things

- $P(y=+1|x,\theta) = \frac{1}{1+e^{-\theta^T x}}$
- $P(y=-1|x,\theta) = 1 - \frac{1}{1+e^{-\theta^T x}} =$
- $\mathbb{1}[y=-1] = 1 - \mathbb{1}[y=+1]$
- Therefore:

$$\max_{\theta} \ell\ell(\theta) = \max_{\theta} \sum_{i=1}^{N} \Big[\mathbb{1}[y=+1] lnP(y^{(i)}=+1|x^{(i)},\theta) + \\ + \mathbb{1}[y=-1] lnP(y^{(i)}=-1|x^{(i)},\theta) \Big] =$$

Simplifying things

$$\max_{\theta} \ell\ell(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{1}[y = +1] ln \frac{1}{1 + e^{-\theta x^{(i)}}} + \mathbb{1}[y = -1] ln \frac{e^{-\theta x^{(i)}}}{1 + e^{-\theta x^{(i)}}} \right] =$$

Calculating gradient

- Log likelihood to maximize:
  $\ell\ell(\theta) = \sum_{i=1}^{N} -(1 - \mathbb{1}[y^{(i)} = +1])\theta^T x^{(i)} - ln(1 + e^{-\theta^T x^{(i)}})$
- Gradient for **one** training example: $\frac{\partial \ell\ell(\theta)}{\partial \theta_j} =$

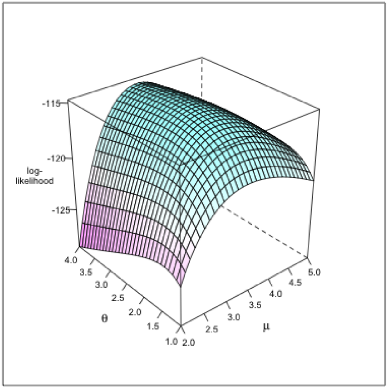
Using gradient

- Gradient:

$$\frac{\partial \ell\ell(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \left(\mathbb{1}[y^{(i)} = +1] - P(y^{(i)} = +1 | \theta, x^{(i)} \right) x_j^{(i)}$$

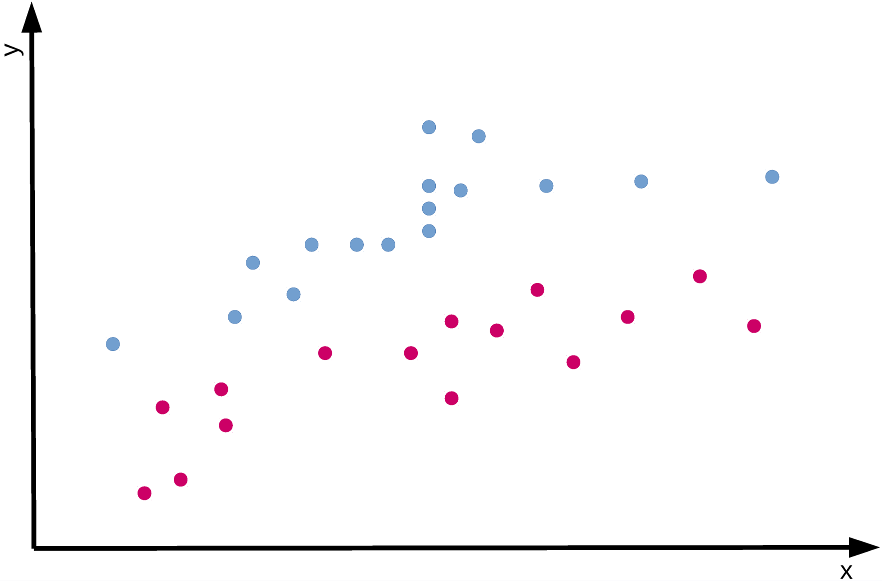
Features

- Log likelihood function is convex, so there is one optimum
- We can use gradient ascent/descent or coordinate ascent/descent without any problems
- We can use Lasso and regularization for linear regression as well

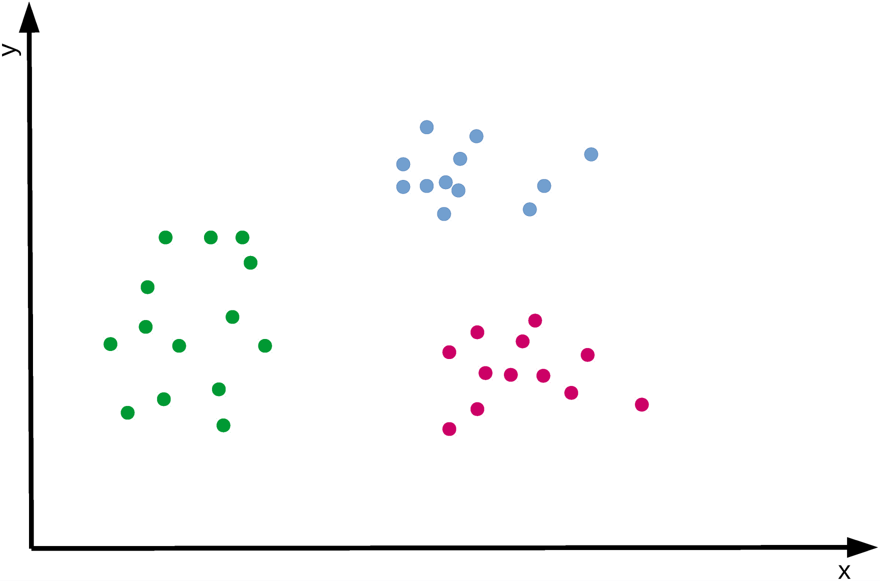


Outline

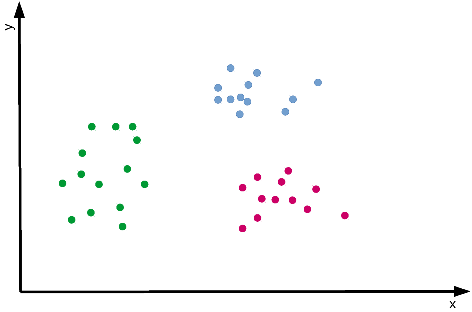
What if we have more than one class?

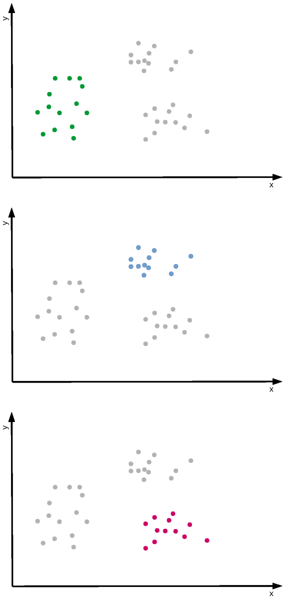


What if we have more than one class?

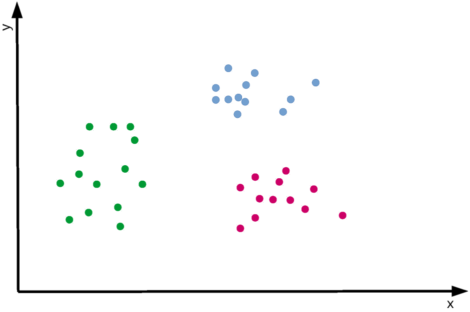


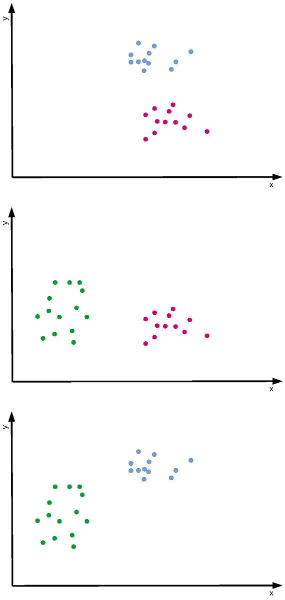
One vs. All





One vs. One





Softmax regression

- We can express this probability in terms of softmax function:

$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{e^{(\theta^{(k)\top} x)}}{\sum_{j=1}^{K} e^{(\theta^{(j)\top} x)}}$$

- Interesting property:

$$\begin{aligned} P(y^{(i)} = k | x^{(i)}; \theta) &= \frac{e^{((\theta^{(k)} - \psi)^\top x^{(i)})}}{\sum_{j=1}^{K} e^{((\theta^{(j)} - \psi)^\top x^{(i)})}} \\ &= \frac{e^{(\theta^{(k)\top} x^{(i)})} e^{(-\psi^\top x^{(i)})}}{\sum_{j=1}^{K} e^{(\theta^{(j)\top} x^{(i)})} e^{(-\psi^\top x^{(i)})}} \\ &= \frac{e^{(\theta^{(k)\top} x^{(i)})}}{\sum_{j=1}^{K} e^{(\theta^{(j)\top} x^{(i)})}}. \end{aligned}$$

Softmax regression

- So in our case:

$$h_\theta(x) = \frac{1}{e^{(\theta^{(1)\top} x)} + e^{(\theta^{(2)\top} x)}} \begin{bmatrix} e^{(\theta^{(1)\top} x)} \\ e^{(\theta^{(2)\top} x)} \end{bmatrix}$$

- And finally:

$$\begin{aligned} h_\theta(x) &= \frac{1}{e^{((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} + e^{(\vec{0}^\top x)}} \begin{bmatrix} e^{((\theta^{(1)}-\theta^{(2)})^\top x)} \\ e^{(\vec{0}^\top x)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{e^{((\theta^{(1)}-\theta^{(2)})^\top x)}}{1+e^{((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}} \\ \frac{1}{1+e^{((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}} \end{bmatrix} \\ &= \begin{bmatrix} 1 - \frac{1}{1+e^{((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}} \\ \frac{1}{1+e^{((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}} \end{bmatrix} \end{aligned}$$

Softmax regression

- Hypothesis:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{(\theta^{(j)\top} x)}} \begin{bmatrix} e^{(\theta^{(1)\top} x)} \\ e^{(\theta^{(2)\top} x)} \\ \vdots \\ e^{(\theta^{(K)\top} x)} \end{bmatrix}$$

- Objective to **maximize**:

$$J(\theta) = \left[\sum_{i=1}^{m} \sum_{k=0}^{1} 1 \left\{ y^{(i)} = k \right\} \ln P(y^{(i)} = k | x^{(i)}; \theta) \right]$$

- Gradient:

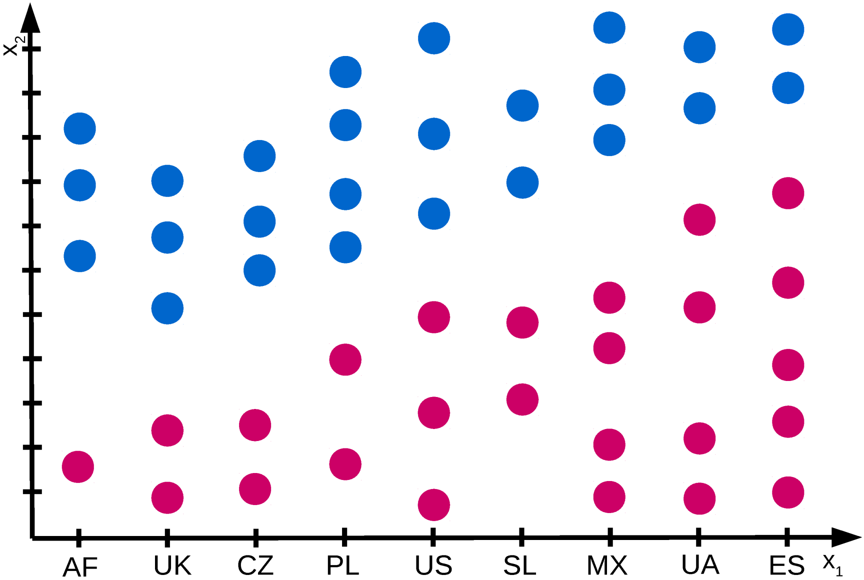
$$\nabla_{\theta^{(k)}} J(\theta) = \sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

Softmax regression – characteristics

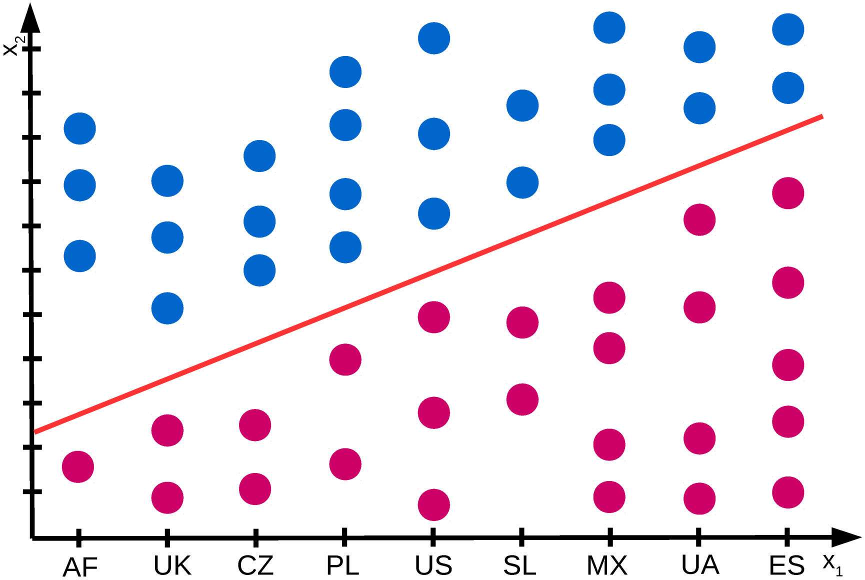
- Convex, so there is no local optima
- Hessian is singular/non-invertible, so only gradient-based optimization is valid
- Returns normalized probability
- Independence assumption between predictions. If the classes are mutually exclusive, use it, otherwise use K-binary classifiers

Outline

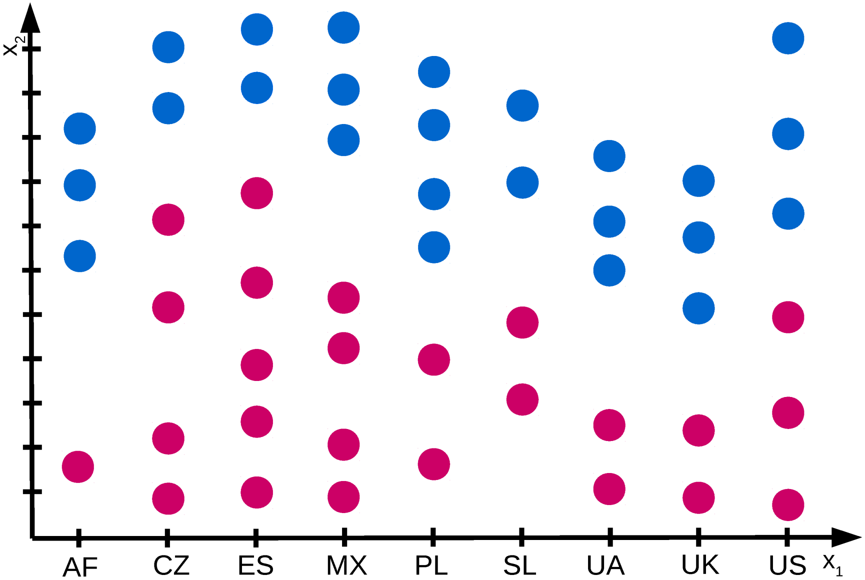
What if we had categorical values?



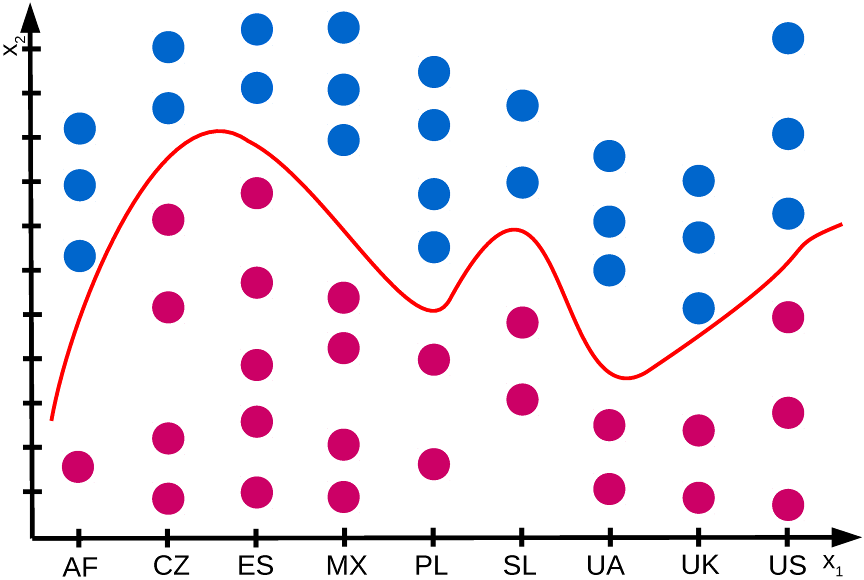
What if we had categorical values?



What if we had categorical values?

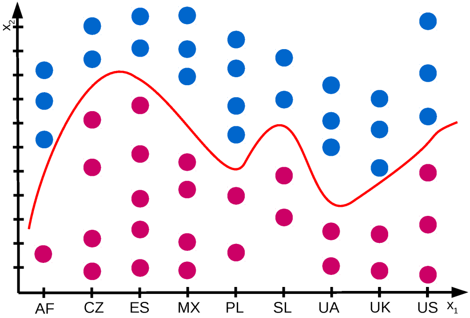


What if we had categorical values?

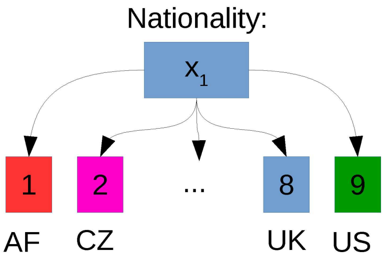


What if we had categorical values?

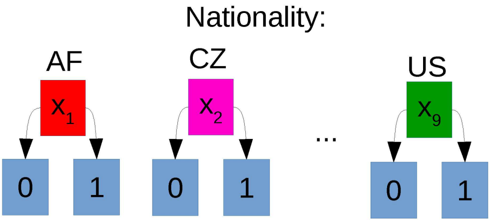
- It is not only a probelm of logistic regression, but also linear regression
- Distance based algorithms also suffer from this problem:
  - Distance from PL to SL is one
  - Distance form PL to CZ is three..
- Soution: one-hot encoding



One-hot encoding



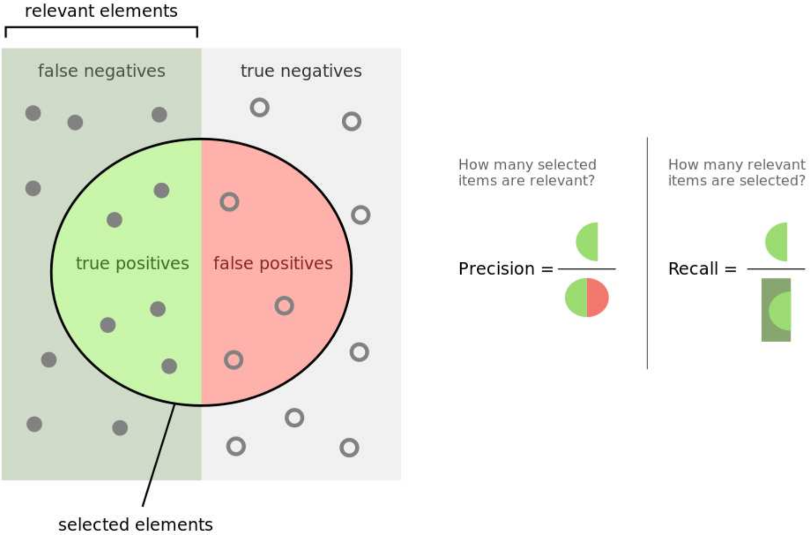
Distance:



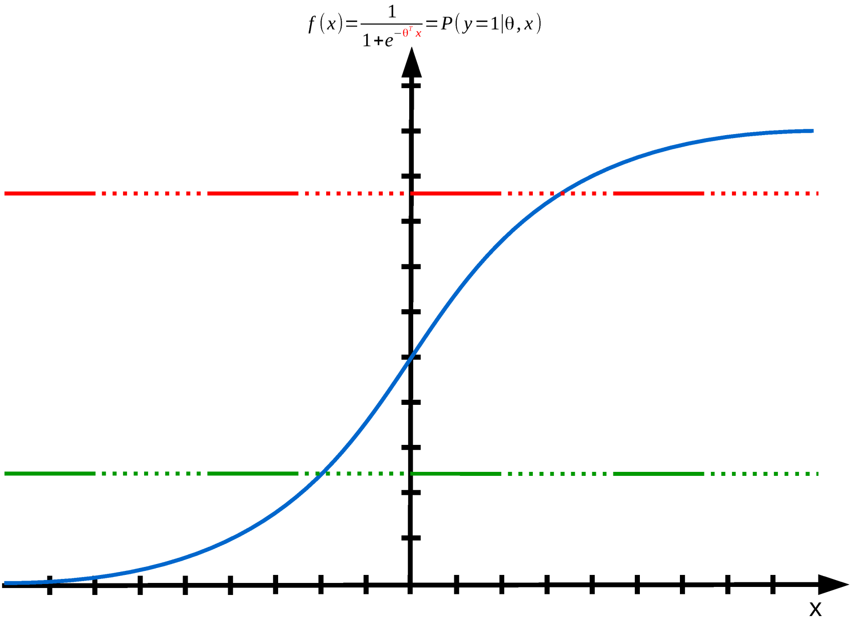
Distance:

Outline

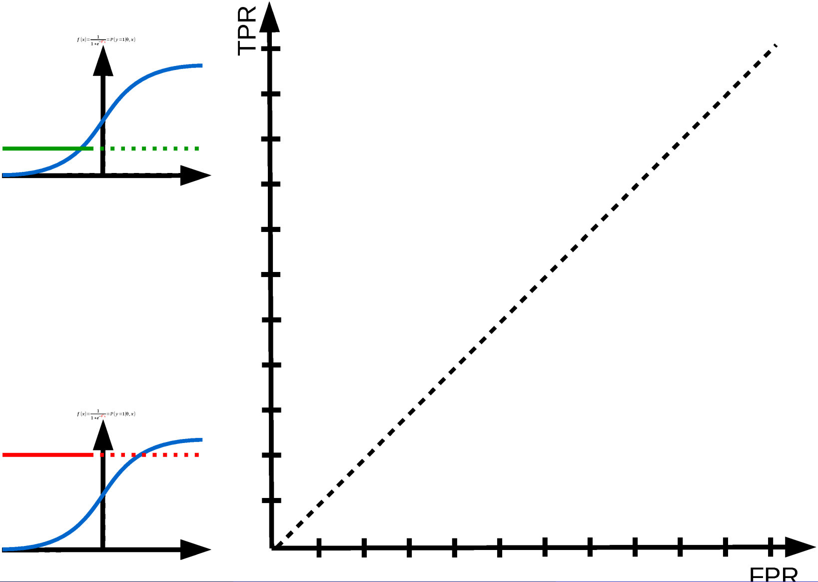
Precision and recall



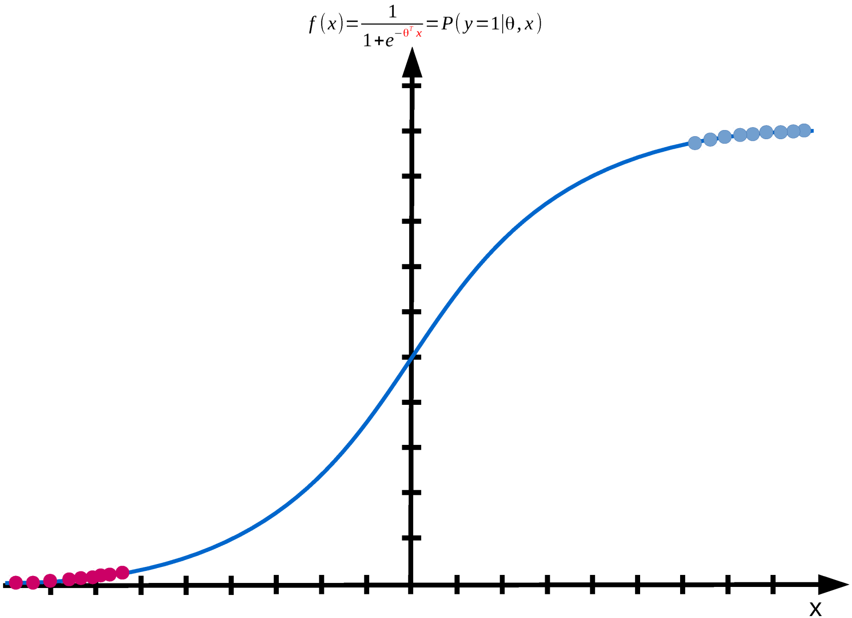
Area under the ROC



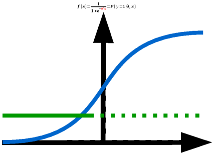
Area under the ROC

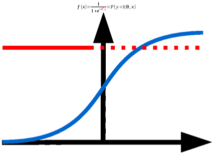


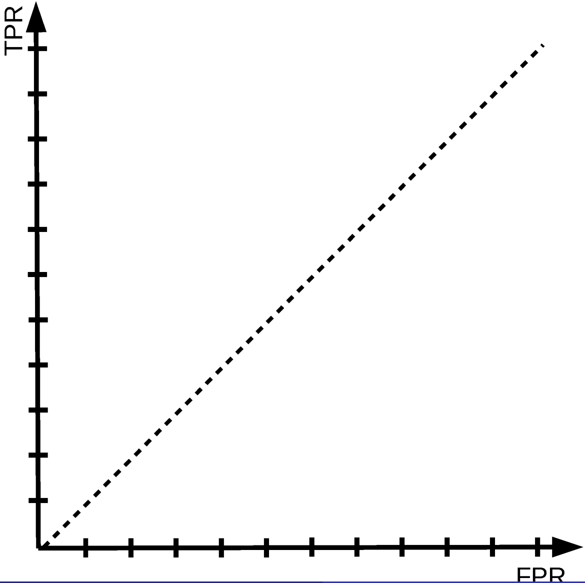
Perfect classifier



Perfect classifier

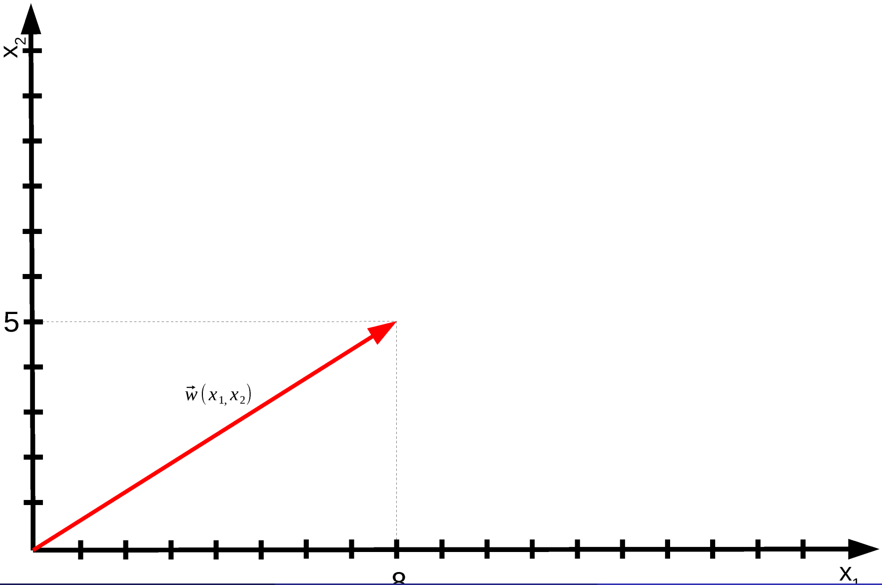






Presentation Outline

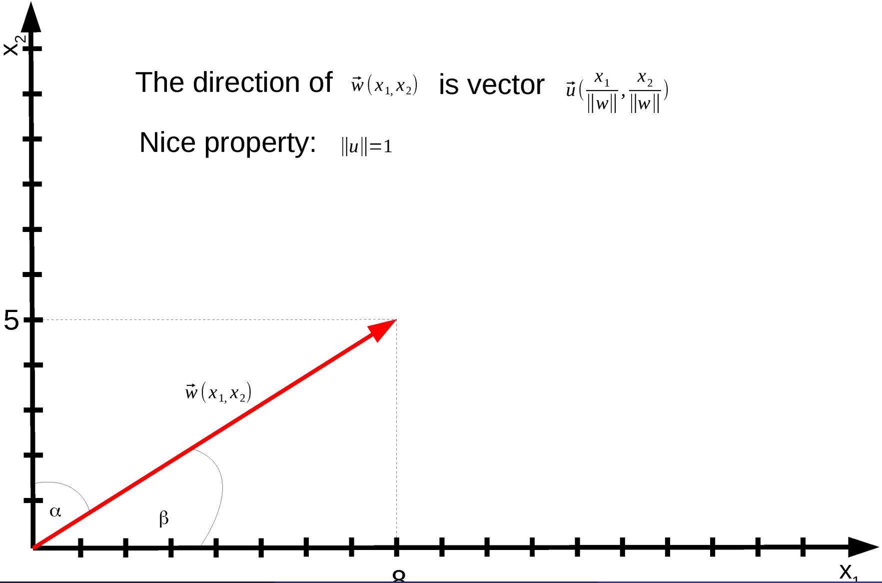
Vector



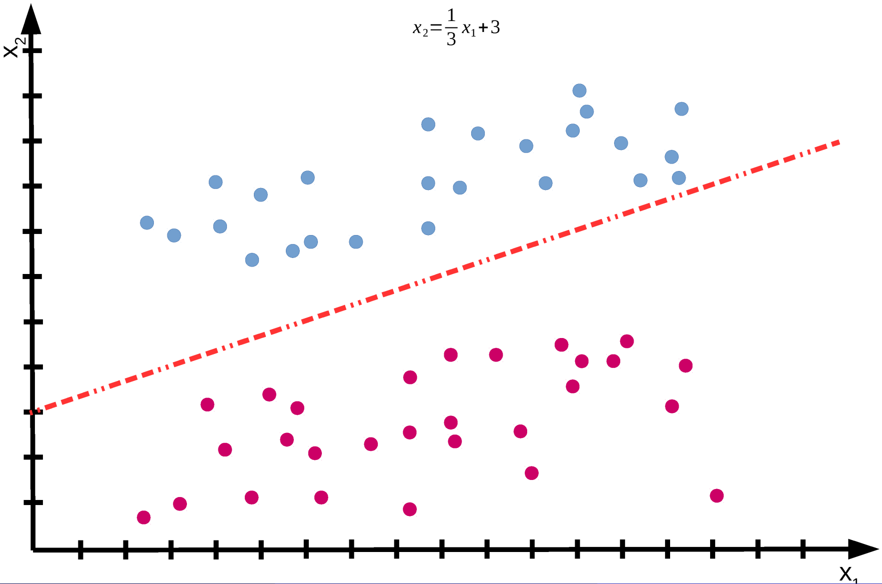
Vector direction

The direction of $\vec{w}(x_1, x_2)$ is vector $\vec{u}\left(\frac{x_1}{\|w\|}, \frac{x_2}{\|w\|}\right)$

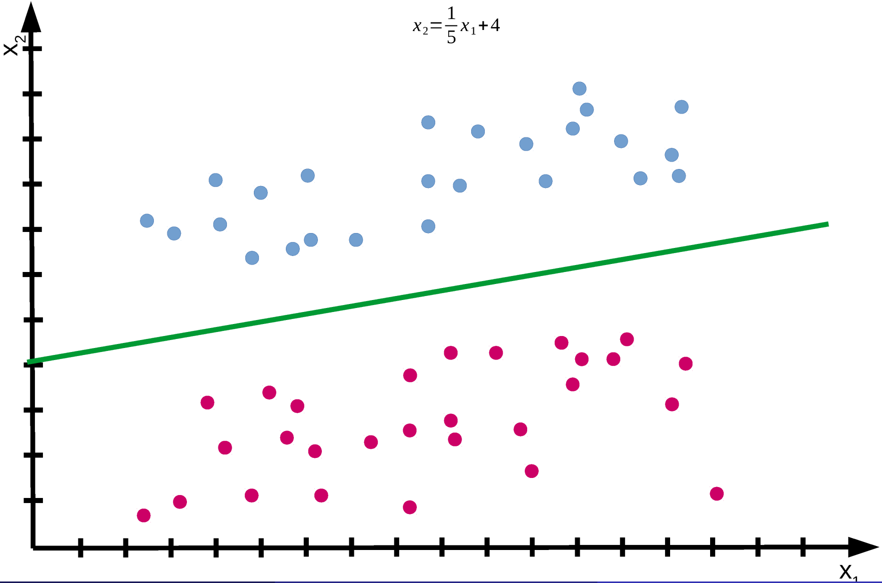
Nice property: $\|u\| = 1$



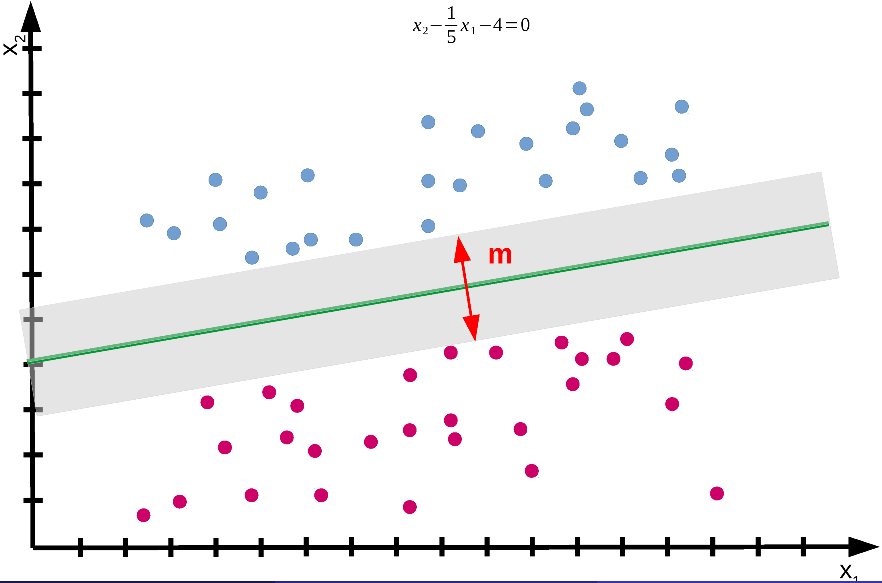
Large margin classifier



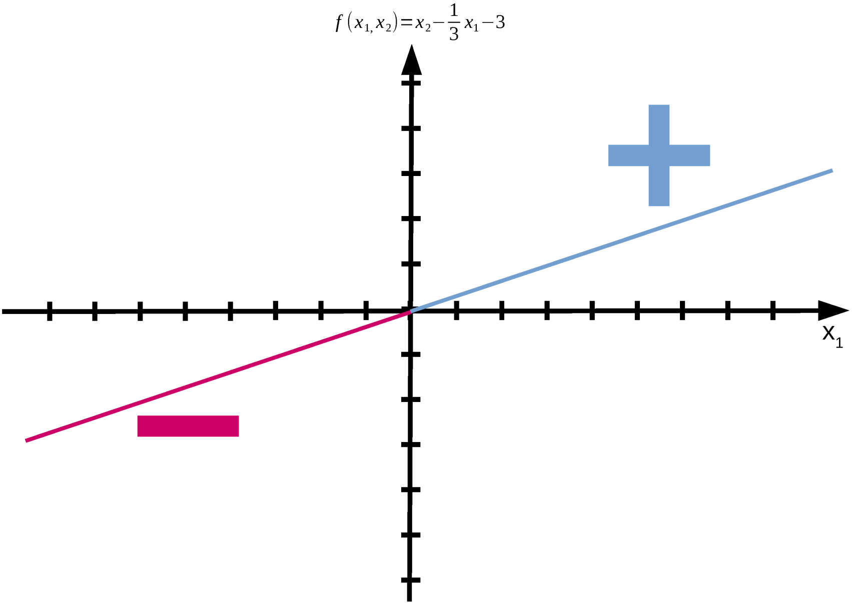
Large margin classifier



Large margin classifier

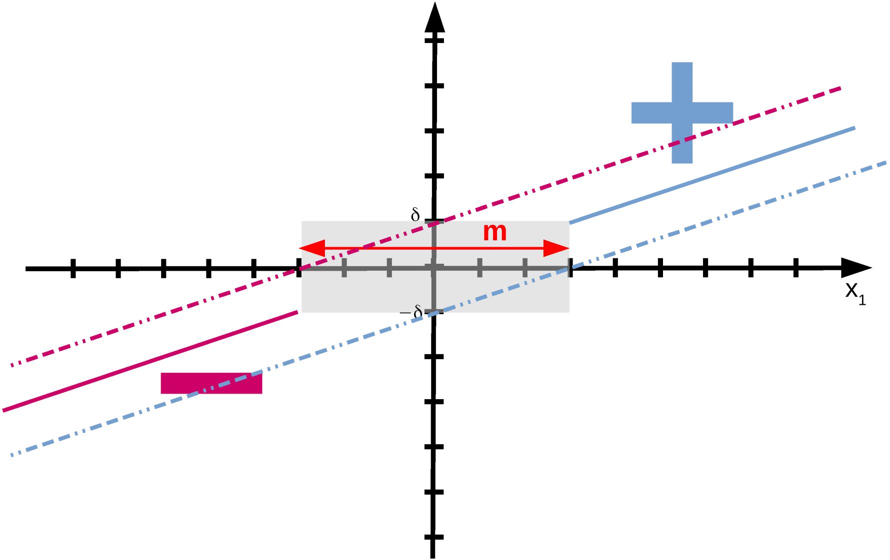


How do we find a margin

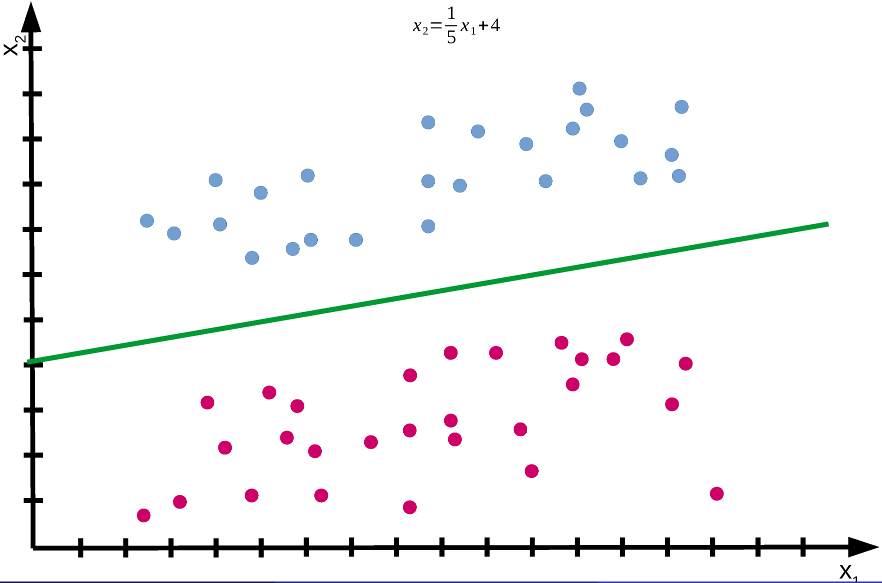


How do we find a margin

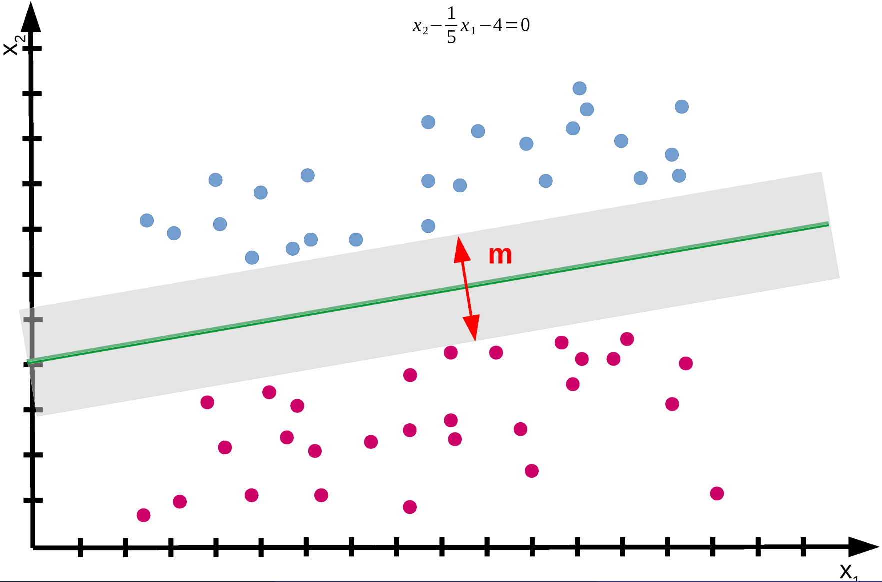
$$f(x_1, x_2) = x_2 - \frac{1}{3}x_1 - 3$$



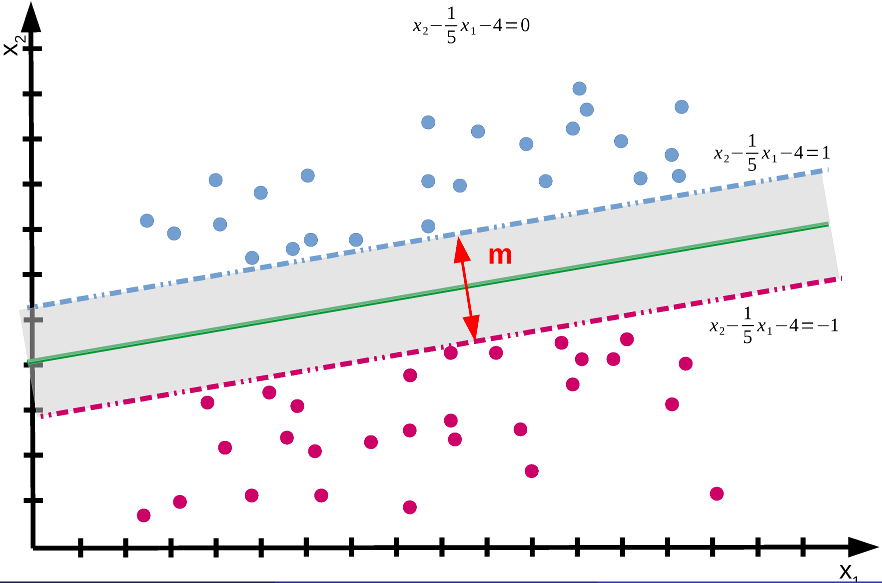
How do we find a margin



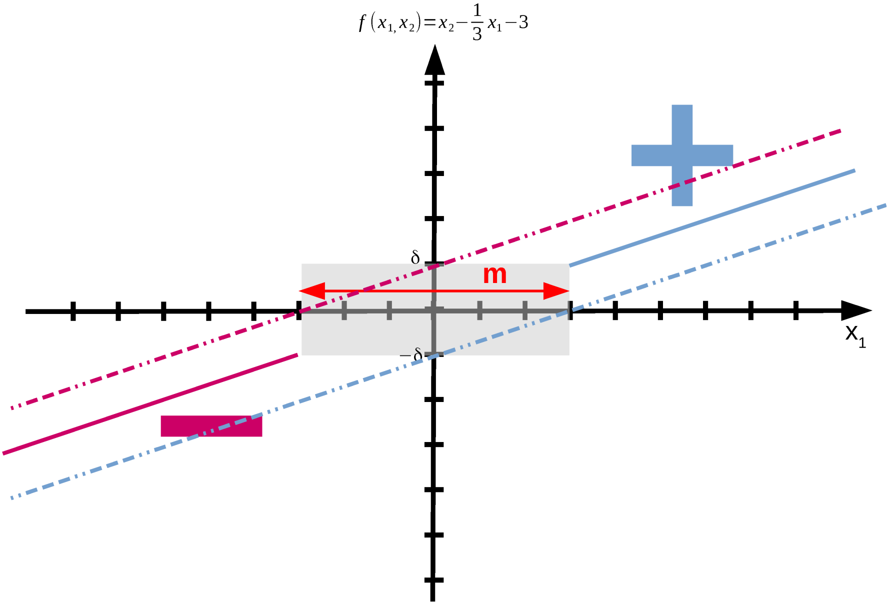
How do we find a margin



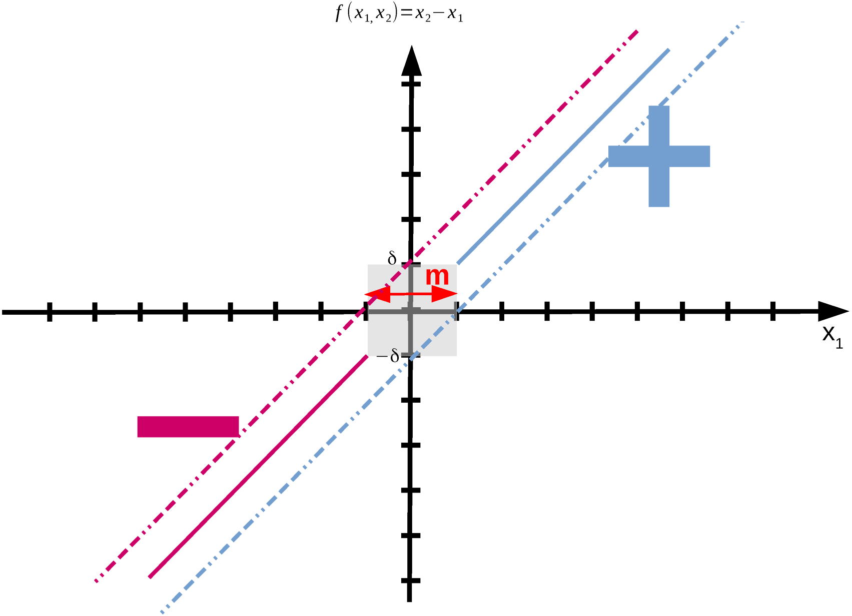
How do we find a margin



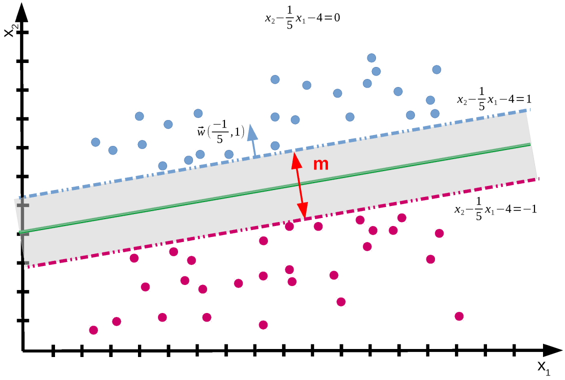
How do we find a margin



How do we find a margin



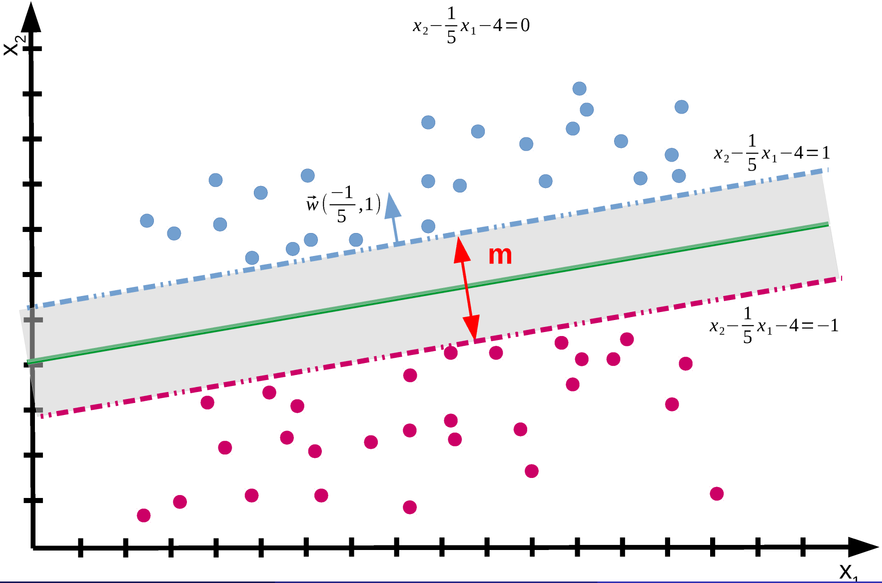
What has an impact on margin



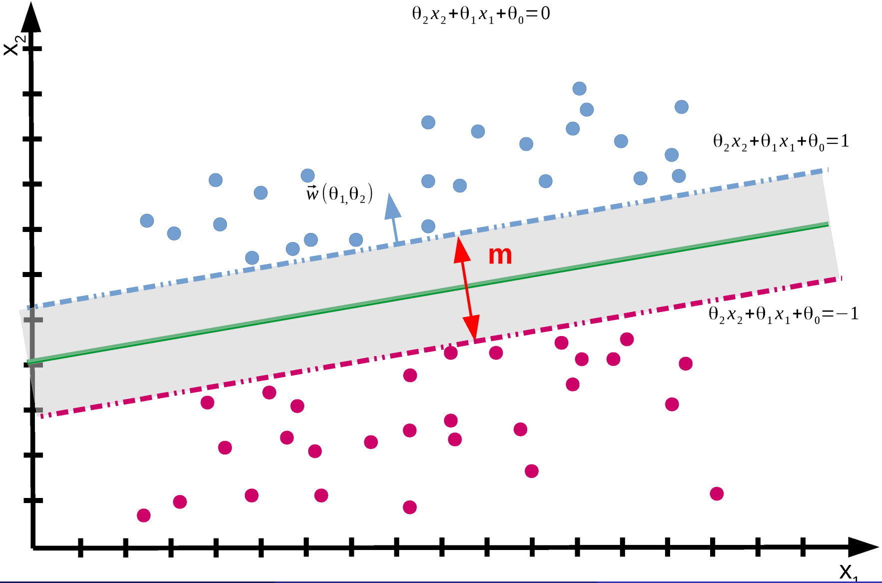
Find magnitude of m

- Vector that defines hyperplane is perpendicular to it (by definition)
- Define direction of that vector (m has the same direction)
- Multiply it by m (direction vectors has norm equal to one)
- Now we have the m vector, and we can calculate its norm.

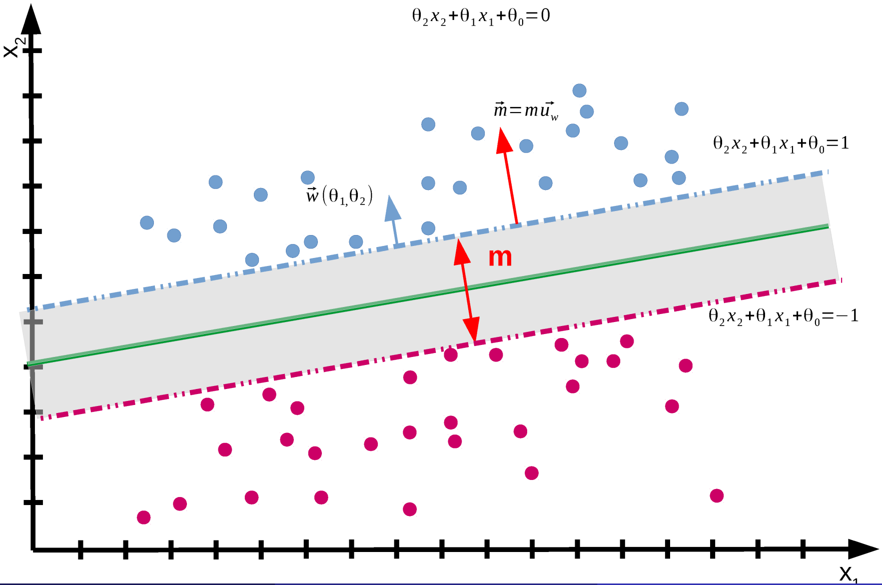
Let us be more general



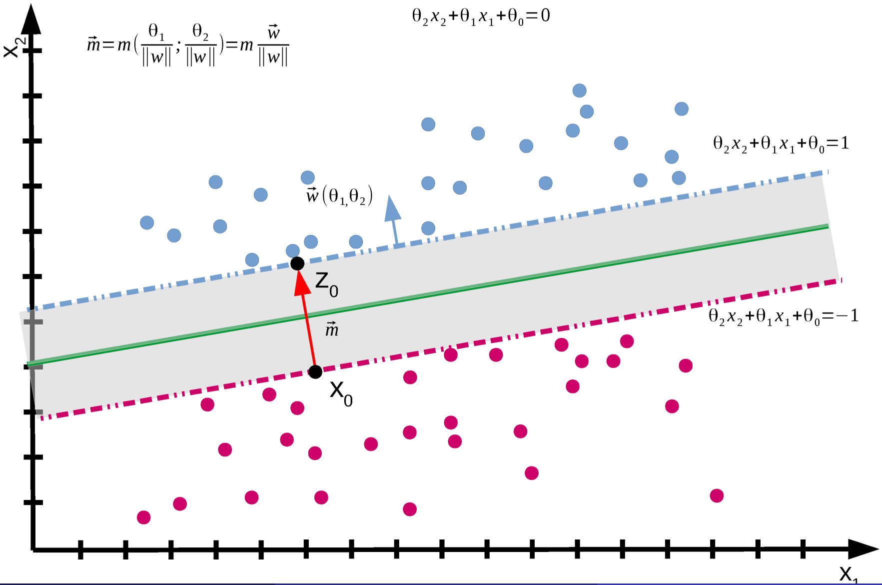
Let us be more general



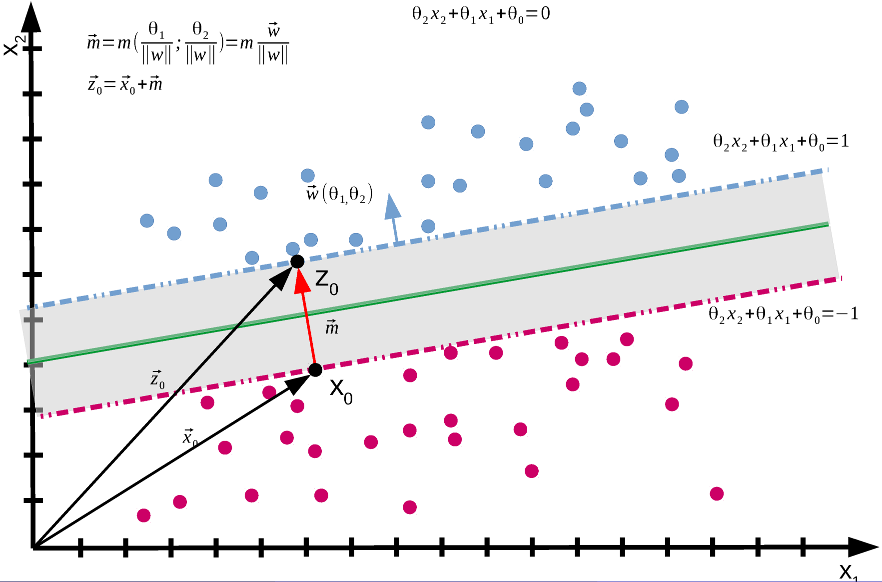
Let us be more general



Let us be more general



Let us be more general



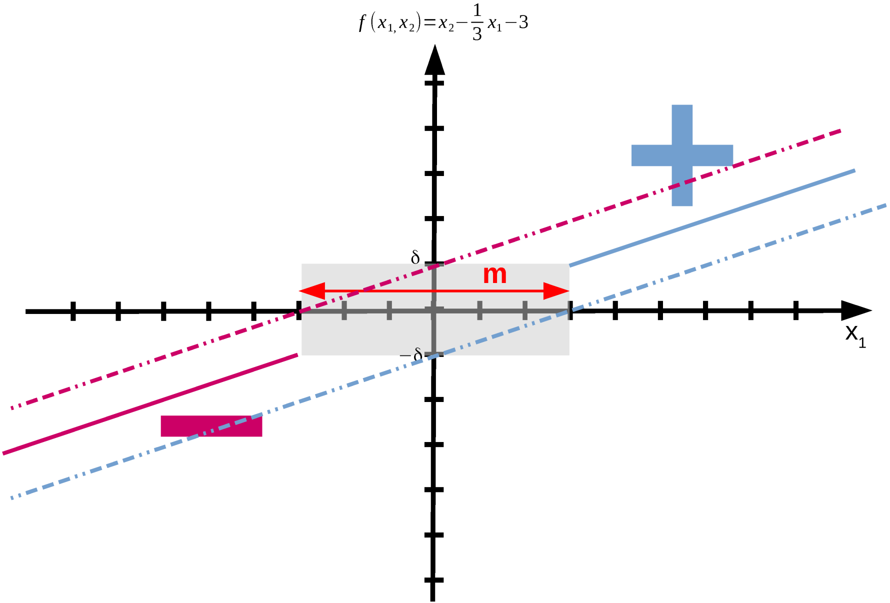
Summing up

- $\vec{m} = m\frac{\vec{w}}{\|w\|}$
- $\vec{z_0} = \vec{x_0} + \vec{m}$
- x_0 belongs to 'upper' hyperplane, so: $\vec{w} \cdot \vec{z_0} + \theta_0 = 1$
- Replace z_0 with: $\vec{w} \cdot (\vec{x_0} + \vec{m}) + \theta_0 = 1$
- Replace $\vec{m}$ with: $\vec{w} \cdot (\vec{x_0} + m\frac{\vec{w}}{\|w\|}) + \theta_0 = 1$
- Expand:

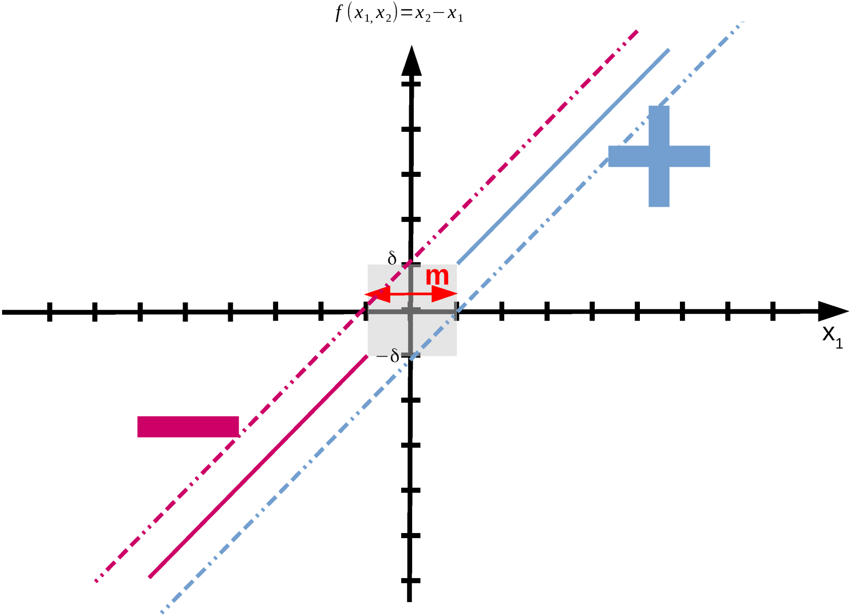
$$
\begin{aligned}
\vec{w} \cdot (\vec{x_0} + m\frac{\vec{w}}{\|w\|}) + \theta_0 &= 1 \\
\vec{w} \cdot \vec{x_0} + m\frac{\vec{w} \cdot \vec{w}}{\|w\|} + \theta_0 &= 1 \\
\vec{w} \cdot \vec{x_0} + m\frac{\|w\|^2}{\|w\|} + \theta_0 &= 1 \\
\underbrace{\vec{w} \cdot \vec{x_0} + \theta_0}_{\text{Lower hyperplane, so } = -1} \qquad + m\|w\| &= 1
\end{aligned}
$$

$$
m = \frac{2}{\|w\|}
$$

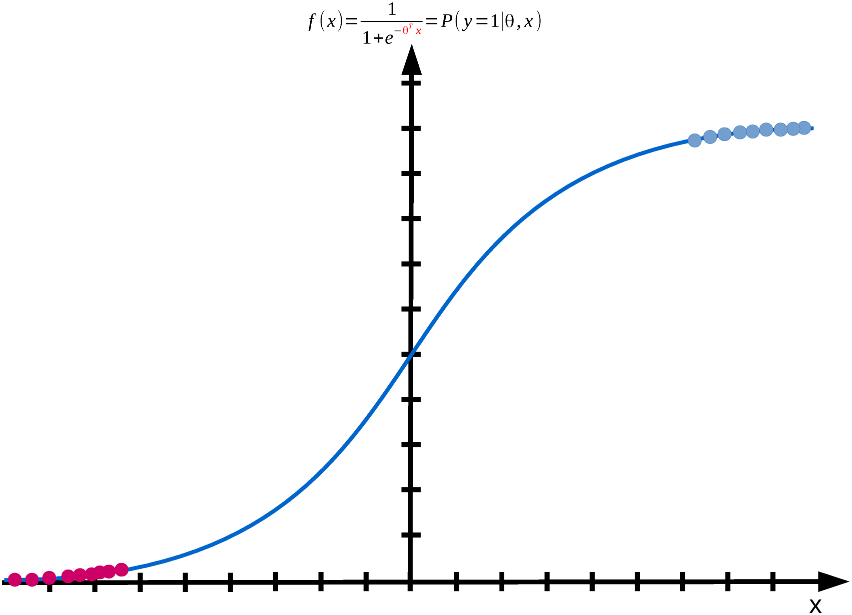
Summing up



Summing up



Wouldn't logistic regression do the same?



Cost function

- To achieve, what we said before, instead of using MSE, or $\ell\ell$, we use hinge loss:

$$\ell(h_\theta(x)) = \max(0, 1 - y \cdot h_\theta(x))$$

  where y is the target label (+1 or -1), and $h_\theta(x)s$ is the predicted label.

- Additionally we add the penalty on margin to cost function. Therefore, the cost function looks as follows:

$$J(\theta) = C \sum_{i}^{N} \max(0, 1 - y^{(i)} \cdot h_\theta(x^{(i)})) + \frac{1}{2}\theta^2$$

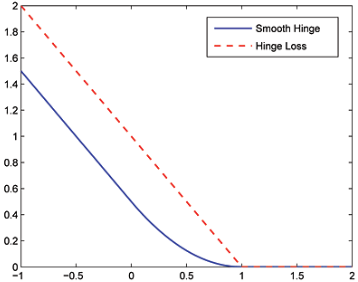
How to optimize cost function

Lagrange multipliers

See: `https://www.svm-tutorial.com/2016/09/duality-lagrange-multipliers/`

Coordinate descent

But, the cost function is not differentiable...

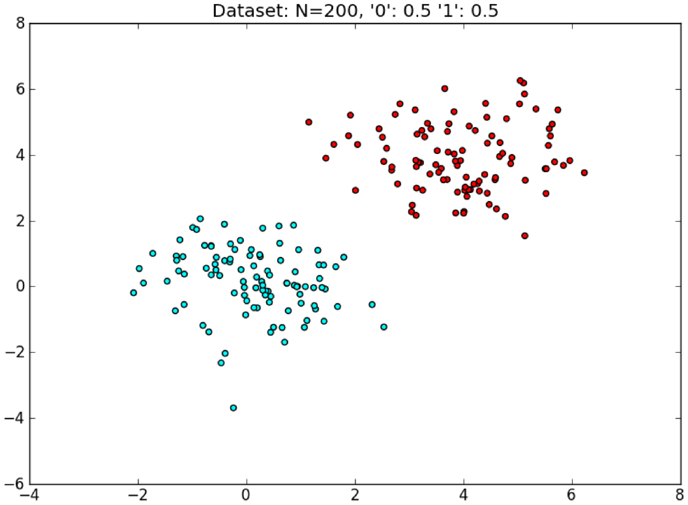


SVM and normalization

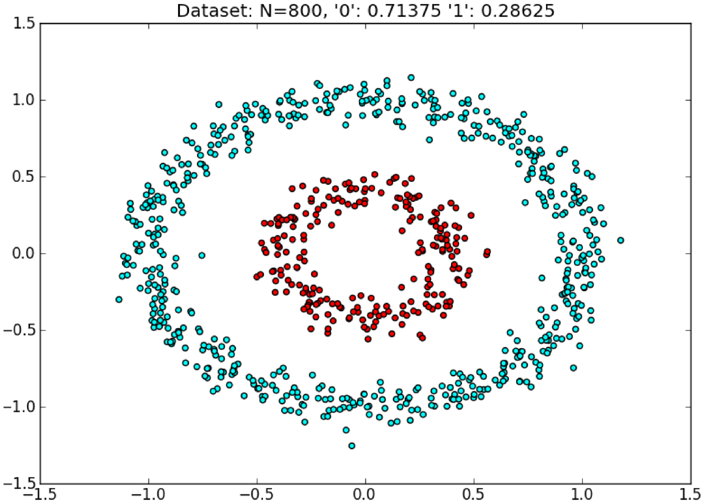
- SVM puts penalty on the value of the θ
- Value of θ depends on the magnitude of gradient
- We multiply each gradient by $x_j^{(i)}$, making θ dependent on the magnitude of $x_j^{(i)}$
- The penalty is therefore dependent on the magnitude of x... :/

Presentation Outline

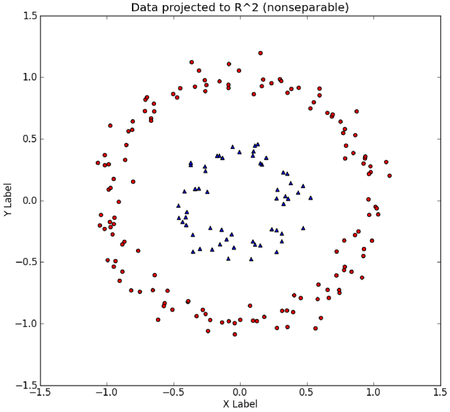
Linearly non separable datasets

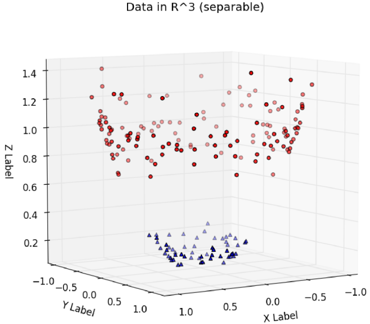


Linearly non separable datasets

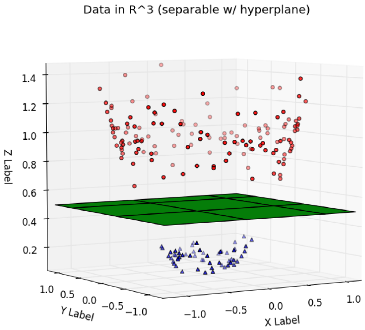


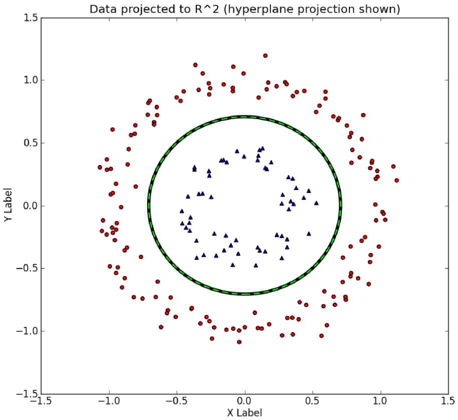
Transformation from lower to higher dimension



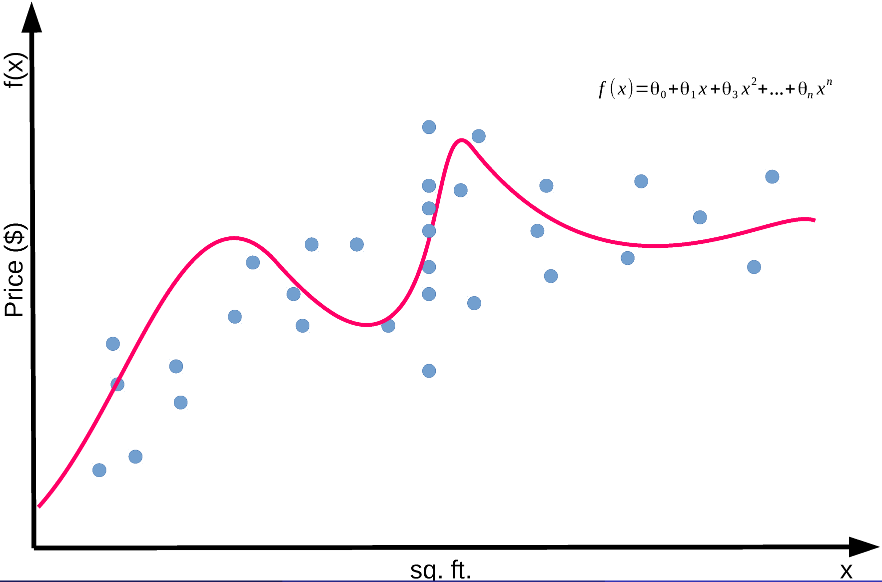


Transformation from lower to higher dimension





We've already done that



Let's move into higher dimension

Getting high is easy ;)

- Let us assume that the original case id 2D $x = (x_1, x_2)$
- Transform $\phi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2)$
- From 2D, we are now in 6D
- Transform $\phi(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$
- From 3D, we are now in 9D
- And so on and on...

There are some consequences, though

- The transform takes resources (both CPU and memory)
- The optimization problem becomes more complex (N dimensions means N θ-s to learn

Outline

Simple Perceptron vs Dual Perceptron

- Imagine binary classification problem, between classes $y \in \{-1, 1\}$
- The classification is performed by the linear model of form: $\widehat{y}(x) = sign(\theta x)$

Algorithm 1: Simple perceptron

```
Data: 𝔻 – dataset of (x, y)
while not converged do
    forall (x^(i), y^(i)) ∈ D do
        if ŷ^(i) y^(i) ≤ 0 then
            θ = θ + λ y^(i) x^(i);
        end
    end
end
```

Conclusion

After the algorithm has converged, we can say how many times each example was misclassified during learning, hence:

$$\theta = \sum_{i}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$$

Simple Perceptron vs Dual Perceptron

- Imagine binary classification problem, between classess $y \in \{-1, 1\}$
- θ can be substituted with $\theta = \sum_i^N \alpha^{(i)} y^{(i)} x^{(i)}$
- The classification is performed by the linear model of form: $\widehat{y}(x) = sign(\theta x)$

Algorithm 2: Dual perceptron

Data: $\mathbb{D}$ – dataset of (x, y)

while ***not converged*** **do**
    forall $(x^{(i)}, y^{(i)}) \in D$ **do**
        if $\widehat{y}^{(i)} y^{(i)} \leq 0$ **then**
            $\alpha = \alpha + 1$;
        end
    end
end

Simple Perceptron vs Dual Perceptron

- Imagine binary classification problem, between classess $y \in \{-1, 1\}$
- θ can be substituted with $\theta = \sum_i^N \alpha^{(i)} y^{(i)} x^{(i)}$
- The classification is performed by the linear model of form: $\widehat{y}(x) = sign\left(\sum_i^N \alpha^{(i)} y^{(i)} x^{(i)} \cdot x\right)$

Algorithm 3: Dual perceptron

Data: $\mathbb{D}$ – dataset of (x, y)

while ***not converged*** **do**
  forall $(x^{(i)}, y^{(i)}) \in D$ **do**
    if $\widehat{y}^{(i)} y^{(i)} \leq 0$ **then**
      $\alpha = \alpha + 1$;
    end
  end
end

Simple SVM vs Dual SVM

- The cost function:

$$J(\theta) = C\sum_{i}^{N} \max(0, 1 - y^{(i)} \cdot h_\theta(x^{(i)})) + \frac{1}{2}\theta^2$$

- Looking at the hinge loss, we can reformulate it in terms of Lagrangian:

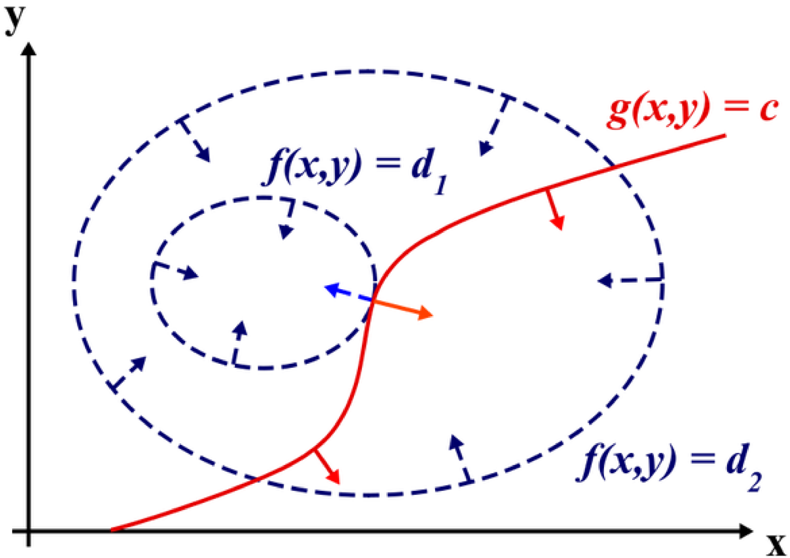
$$J(\theta) = \frac{1}{2}\theta^T\theta \qquad \text{s.t.} \qquad y^{(i)}h_\theta(x^{(i)}) \geq 1 \text{ for } i \in 1 \dots N$$

$$J(\theta) = \frac{1}{2}\theta^T\theta \qquad \text{s.t.} \qquad y^{(i)}(\theta x^{(i)} + b) \geq 1 \text{ for } i \in 1 \dots N$$

- Now, substitute to Lagrange function:

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

Simple SVM vs Dual SVM



Derivatives of Lagrangian

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

- With respect to θ:

$$\nabla_\theta \mathcal{L} = \theta - \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$$

- So setting gradient to 0, we have:

$$\theta = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$$

- With respect to b:

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{N} \alpha^{(i)} y^{(i)}$$

- So setting gradient to 0, we have:

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$$

Putting it altogether

What we have

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

- $\theta = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

$$\mathcal{L}(\theta, b, \alpha) = \sum_{i=1}^{N} \alpha^{(i)} -$$

Putting it altogether

What we have

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

- $\theta = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

$$\mathcal{L}(\theta, b, \alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} x^{(i)T} x^{(j)}$$

Putting it altogether

What we have

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

- $\theta = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} x^{(i)T} x^{(j)}$$

Putting it altogether

What we have

$$\mathcal{L}(\theta, b, \alpha) = \frac{1}{2}\theta^T\theta - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(\theta x^{(i)} + b) - 1 \right]$$

- $\theta = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} x^{(i)T} x^{(j)}$$

Subjected to:

- $\alpha^{(i)} \geq 0$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

Using it with higher dimensions

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} x^{(i)T} x^{(j)}$$

Subjected to:

- $\alpha^{(i)} \geq 0$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

Using it with higher dimensions

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} z^{(i)}, z^{(j)}$$

Subjected to:

- $\alpha^{(i)} \geq 0$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

Wrap up

What did we learn

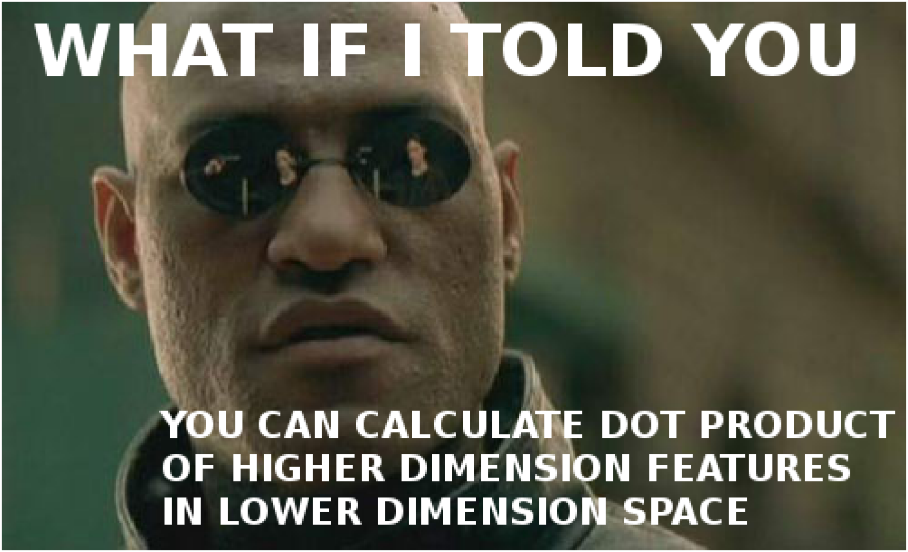
- We know that we can represent our optimization problem in terms of dot product of training examples, not θ.
- We know that dot product is easy to compute.
- We know, that finding non linear decision boundary is possible by transforming feature space to higher dimension.
- On the other hand we know, that moving into higher dimension is bad.
- **So what did we learn?**

Wrap up

What did we learn

- We know that we can represent our optimization problem in terms of dot product of training examples, not θ.
- We know that dot product is easy to compute.
- We know, that finding non linear decision boundary is possible by transforming feature space to higher dimension.
- On the other hand we know, that moving into higher dimension is bad.
- **So what did we learn?**
- **We probably could do in the future better if we only knew what we did :)**

The kernel trick



Example

- For transform:

$$\phi(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$$

  we have the following kernel:

$$K(x, x') = (x \cdot x')^2$$

- Example. Assume $x^{(1)} = (1, 2, 3)$ and $x^{(2)} = (4, 5, 6)$

$$\begin{aligned}
\phi(x^{(1)}) &= (1, 2, 3, 2, 4, 6, 3, 6, 9) \\
\phi(x^{(2)}) &= (16, 20, 24, 20, 25, 30, 24, 30, 36) \\
\left\langle \phi(x^{(1)}), \phi(x^{(1)}) \right\rangle &= 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 \\
&= 1024
\end{aligned}$$

- Kernel:

$$K(x^{(1)}, x^{(2)}) = (4 + 10 + 18)^2 = 32^2 = 1024$$

Example

- For transform
$$\phi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2)$$
we have the following kernel:
$$K(x, x') = (1 + x^T x')^2$$
- Example. Assume $x^{(1)} = (1, 2)$ and $x^{(2)} = (3, 4)$
$$\begin{aligned}\phi(x^{(1)}) &= (1, 1, 4, 1\sqrt{2}, 2\sqrt{2}, 1 \cdot 2\sqrt{2})\\ \phi(x^{(2)}) &= (1, 9, 16, 3\sqrt{2}, 4\sqrt{2}, 12\sqrt{2})\\ \left\langle \phi(x^{(1)}), \phi(x^{(1)}) \right\rangle &= 1 + 9 + 64 + 6 + 16 + 48 = 144\\ &= 1024\end{aligned}$$
- Kernel:
$$K(x^{(1)}, x^{(2)}) = (1 + 3 + 8)^2 = 12^2 = 144$$

How far can we go?

- Let us take Gaussian kernel: $K(x, x') = e^{-\gamma \|x - x'\|^2}$
- e can be expanded into Taylor series: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$
- So, let's assume $\gamma = \frac{1}{2}$, expand and substitute into Taylor series:

$$
\begin{aligned}
e^{-\frac{1}{2}\|x - x'\|^2} &= e^{-x^2 + \langle x, x' \rangle - x'^2} \\
&= e^{-x^2} e^{x'^2} \sum_{k}^{\infty} \frac{\langle x, x' \rangle^k}{k!}
\end{aligned}
$$

- So the transform function is (1D case):

How far can we go?

- Let us take Gaussian kernel: $K(x, x') = e^{-\gamma \|x - x'\|^2}$
- e can be expanded into Taylor series: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$
- So, let's assume $\gamma = \frac{1}{2}$, expand and substitute into Taylor series:

$$e^{-\frac{1}{2}\|x-x'\|^2} = e^{-x^2 + \langle x, x' \rangle - x'^2}$$
$$= e^{-x^2} e^{x'^2} \sum_{k}^{\infty} \frac{\langle x, x' \rangle^k}{k!}$$

- So the transform function is (1D case):

$$\phi(x) = \left[e^{-x^2}, \sqrt{\frac{e^{-x^2}}{1!}} x, \sqrt{\frac{e^{-x^2}}{2!}} x^2, \sqrt{\frac{e^{-x^2}}{3!}} x^3, \dots \right]$$

How far can we go?

Using it in dual representation

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} x^{(i)T} x^{(j)}$$

Subjected to:

- $\alpha^{(i)} \geq 0$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

Using it in dual representation

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} K(x^{(i)}, x^{(j)})$$

Subjected to:

- $\alpha^{(i)} \geq 0$
- $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$

Constructing kernels

- Not every function is a kernel (see Mercer's theorem). To be one, it has to be:
  1. symmetric: $K(x,x') = K(x',x)$
  2. positive semidefinite: $\sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i,x_j)c_i c_j \geq 0$
- New kernels can be constructed as combination of already known kernels:

> Techniques for Constructing New Kernels.
>
> Given valid kernels $k_1(\mathbf{x},\mathbf{x}')$ and $k_2(\mathbf{x},\mathbf{x}')$, the following new kernels will also be valid:
>
> $$k(\mathbf{x},\mathbf{x}') = ck_1(\mathbf{x},\mathbf{x}') \quad (6.13)$$
> $$k(\mathbf{x},\mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}') \quad (6.14)$$
> $$k(\mathbf{x},\mathbf{x}') = q(k_1(\mathbf{x},\mathbf{x}')) \quad (6.15)$$
> $$k(\mathbf{x},\mathbf{x}') = \exp(k_1(\mathbf{x},\mathbf{x}')) \quad (6.16)$$
> $$k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}') \quad (6.17)$$
> $$k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}') \quad (6.18)$$
> $$k(\mathbf{x},\mathbf{x}') = k_3(\phi(\mathbf{x}),\phi(\mathbf{x}')) \quad (6.19)$$
> $$k(\mathbf{x},\mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \quad (6.20)$$
> $$k(\mathbf{x},\mathbf{x}') = k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b) \quad (6.21)$$
> $$k(\mathbf{x},\mathbf{x}') = k_a(\mathbf{x}_a,\mathbf{x}'_a)k_b(\mathbf{x}_b,\mathbf{x}'_b) \quad (6.22)$$
>
> where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot,\cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a,\mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Thank you!

Szymon Bobek
Institute of Applied Computer Science
AGH University of Science and Technology
21 March 2017

http://geist.agh.edu.pl