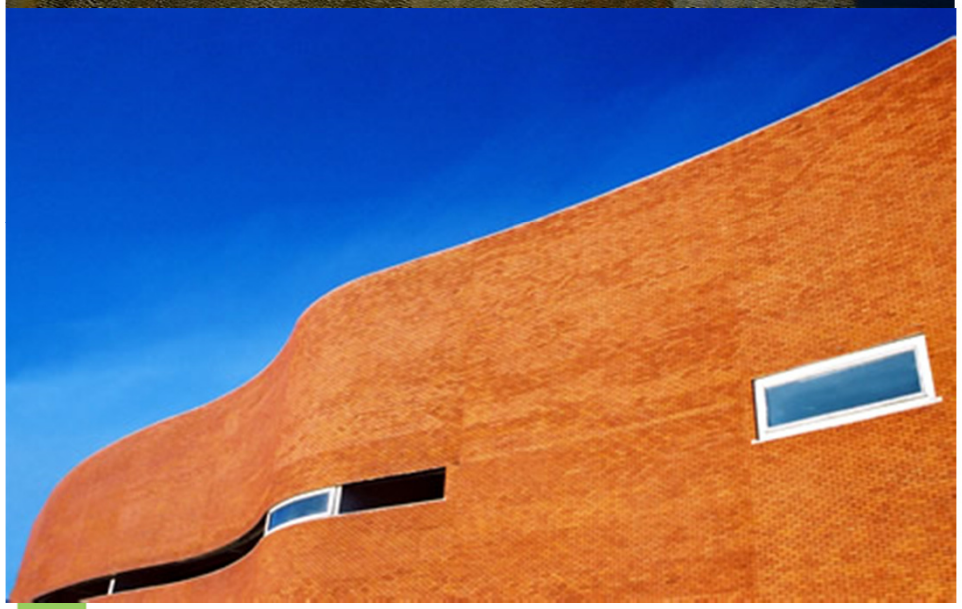
Aprendizagem Aplicada à Segurança

(Mestrado em Cibersegurança-DETI-UA)



LECTURE 2
Supervised learning – Classification

Petia Georgieva
(petia@ua.pt)

DETI/IEETA – UA

OUTLINE

- **Logistic Regression (logit model)**
- **Support Vector machines (SVM)**
- **K- Nearest-Neighbor (k-NN)**

Classification –

LOGISTIC REGRESSION (LOGIT)

Binary vs Multiclass Classification

Email: Spam / NOT Spam?
Online Bank Transaction: Fraudulent (Yes /No) ?
Network traffic : OK/Malware

Binary classification:
y = 1: “positive class” (e.g. Malware traffic)
y = 0: “negative class” (e.g. OK traffic)

Find a model h(x) that outputs values between 0 and 1
0<=h(x)<=1
if h(x) >=0.5, predict “y=1”
if h(x) <0.5, predict “y=0”

Multiclass classification (*K* classes) => y= {0, 1, 2,..}
Build *K* binary classifiers, for each classifier one of the classes has label 1 all other classes take label 0 .

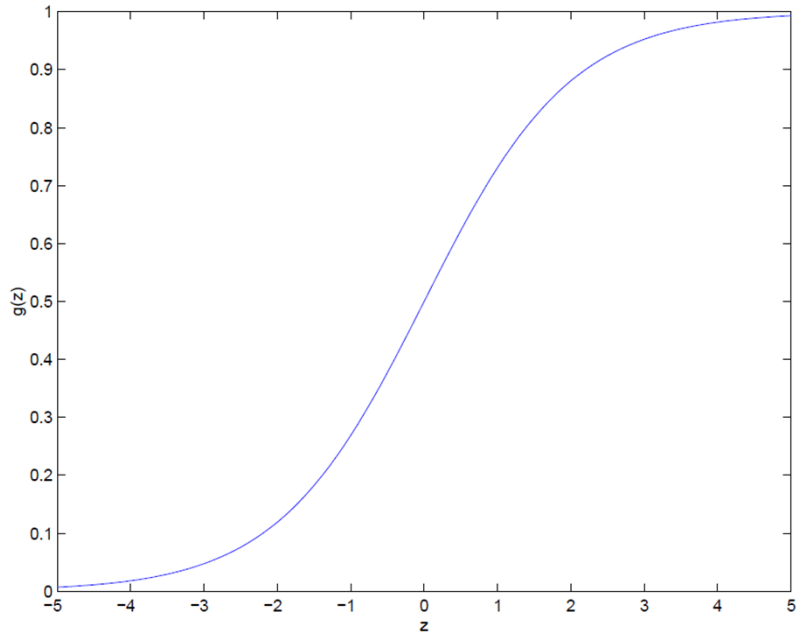
Logistic Regression

Given labelled data of m examples, n features
Labels $\{0,1\}$ => binary classification
x –vector of features; θ – vector of model parameters;
$h(x)$ – logistic (sigmoid function) model – logit model

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1+e^{-z}} = g(\theta^T x) = g(z)$$

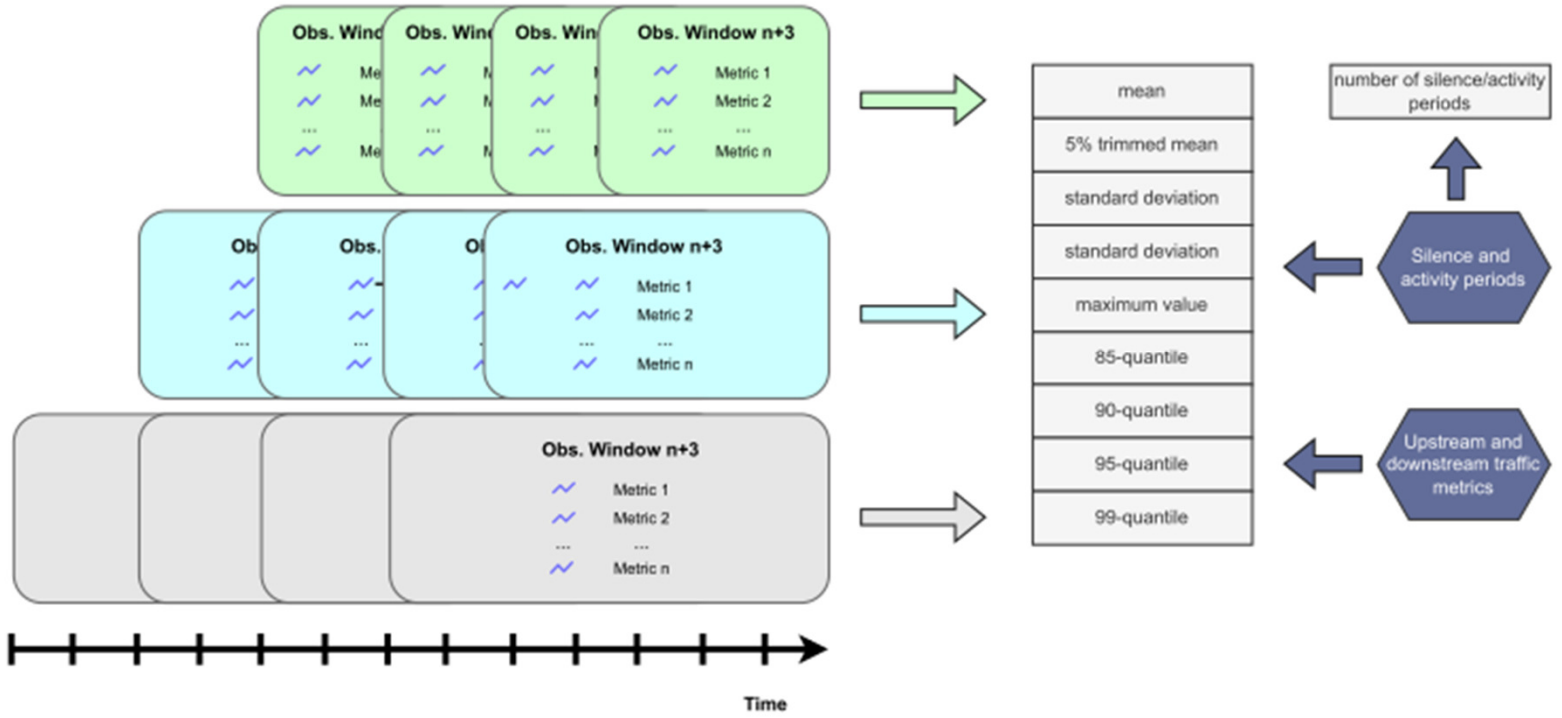
$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots . \theta_n x_n$$

Logistic (sigmoid) function



universidade de aveiro

Feature vector x



Row network data:
collected upstream/downstream network traffic metrics:
uploaded packets (#, Bytes), downloaded packets (#, Bytes), silence/activity periods

Feature vector x:
mean, max, min, standard deviation, different quantiles, over multiple sub-windows

Label y: Network traffic OK (0) / Malware (1)

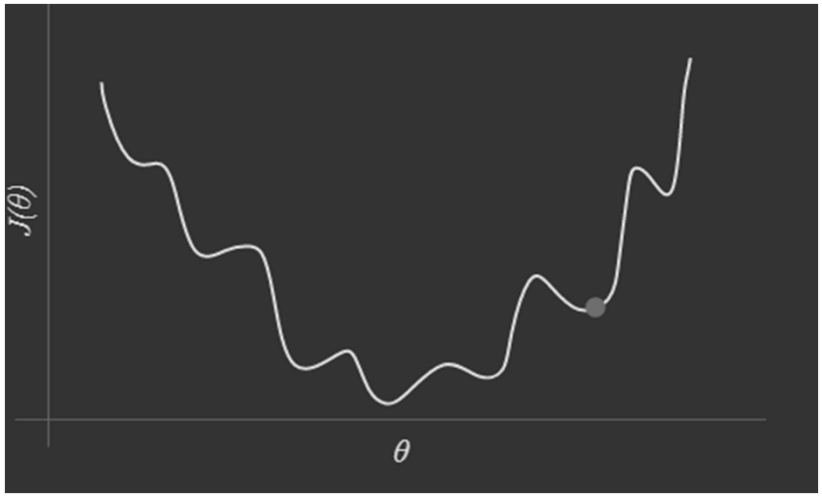
Logistic Regression Cost Function

Linear regression model => $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots . \theta_n x_n = \vec{\theta}^T \vec{x}$

Lin Regr. cost (loss) function (MSE) => $J = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2$

Nonlinear logit model => $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

If we use the same cost function as with linear regression, but now we have the nonlinear logit model, $J(\theta)$ will be a non-convex function (has many local minima)=> **not efficient for optimization !**



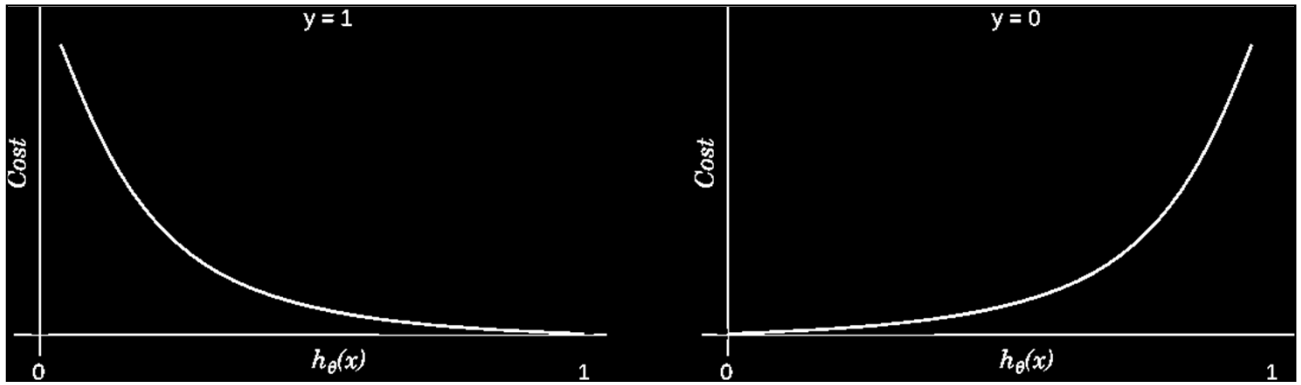
Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

$-\log(h_\theta(x))$ $\qquad$ $-\log(1-h_\theta(x))$



LogReg cost function combined into one expression :
(also known as binary Cross-Entropy or Log Loss function)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

LogReg with gradient descent learning

Inicialize model parameters *(e.g. $\theta = 0$)*
Repeat until J converge {

Compute Logit Model prediction =>
(different from linear regression model)

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Compute Logit cost function =>
(different from linear regression cost function)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Goal =>

$$\min_\theta J(\theta)$$

Compute cost function gradients =>
(same as linear regression gradients)

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Update parameters =>
(same as linear regression parameter update)

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Optimization algorithms

Gradient descent (learned in class) -
updates the parameters in direction in which the gradient decreases most rapidly.

2. Other optimization algorithms
   - Conjugate gradient
   - AdaGrad, RMSProp
   - Stochastic gradient descent with momentum
   - ADAM (combination of RMSProp and stochastic optimization)
   - BFGS (Broyden–Fletcher–Goldfarb–Shanno)
   - Quasi-Newton methods (approximate the second derivative)

Characteristics

- Adaptive learning rate (alfa);
- Often faster than gradient descent; better convergence;
- Approximate (estimate) the true gradient over a mini-batch and not over the whole data;
- More complex algorithms

Logistic regression - example

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0 \Rightarrow \text{decision boundary}$$

$$if \quad z > 0 \Rightarrow g(z) > 0.5 \Rightarrow \text{predict class} = 1$$

$$if \quad z < 0 \Rightarrow g(z) < 0.5 \Rightarrow \text{predict class} = 0$$

$$g(z) = \frac{1}{1+e^{-z}}$$

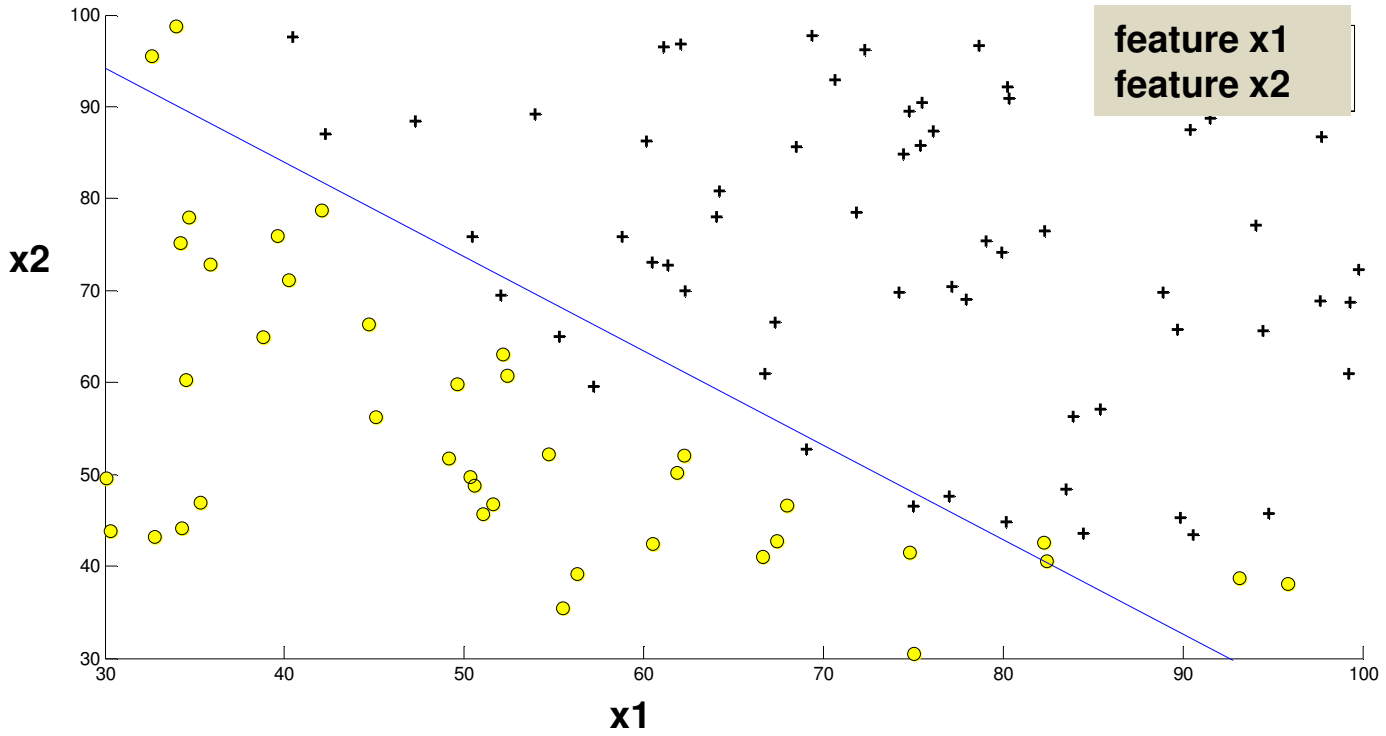
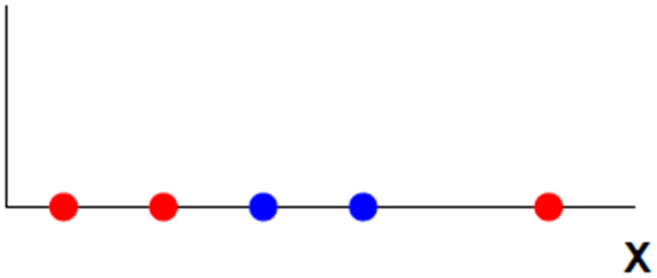


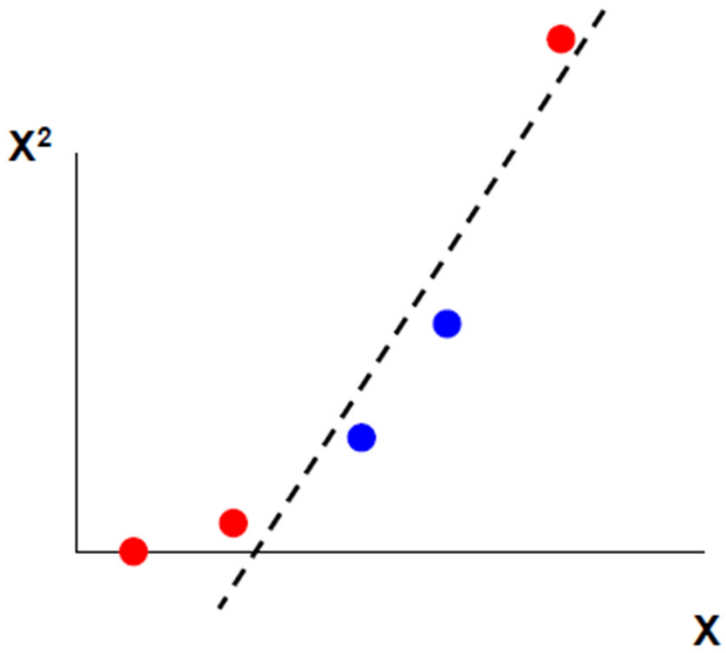
Fig. Training data and linear decision boundary with the optimized parameters (θ)

Nonlinearly Separable Data

Linear classifier cannot classify these examples.



And now ?



$z = \theta^T x = \theta_0 + \theta_1 x + \theta_2 x^2 = 0 =>$

Nonlinear decision boundary (in the original feature space x)

Linear decision boundary (in the extended feature space x, x^2)

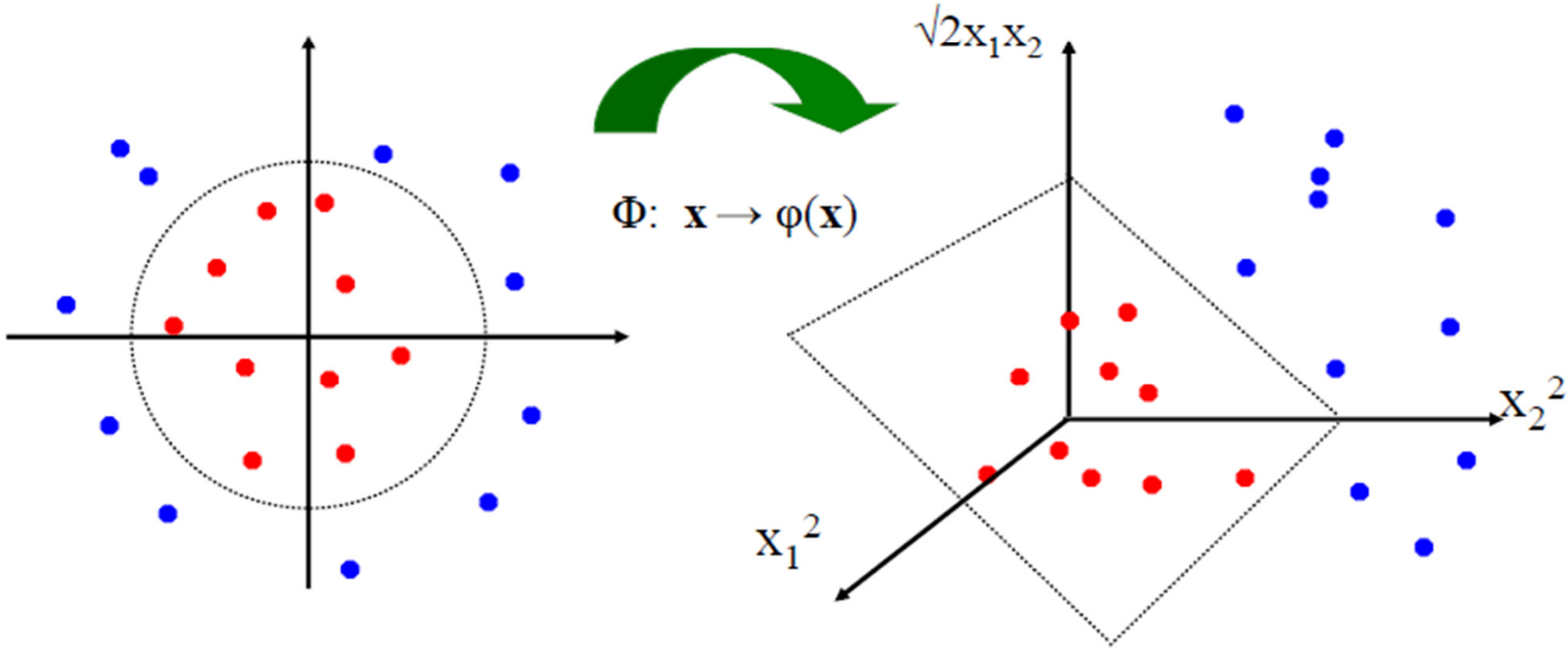
if $z > 0 => g(z) > 0.5 =>$ predict class = 1

if $z < 0 => g(z) < 0.5 =>$ predict class = 0

Nonlinearly Separable Data

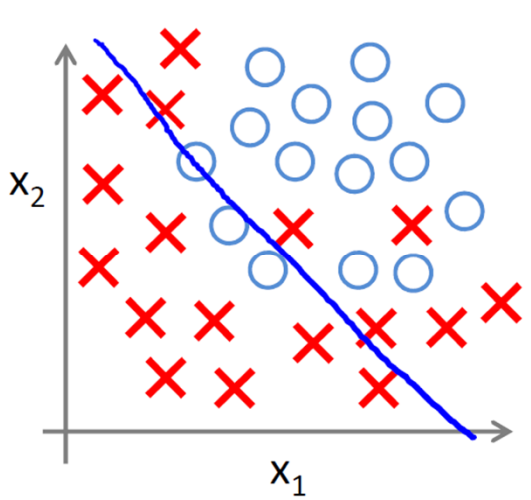
- The original input space (x) can be mapped to some higher-dimensional feature space ($\varphi(\mathbf{x})$)where the training set is separable:

$x=(x_1,x_2)$ $\varphi(\mathbf{x}) =(x_1^2,x_2^2,\sqrt{2}x_1x_2)$

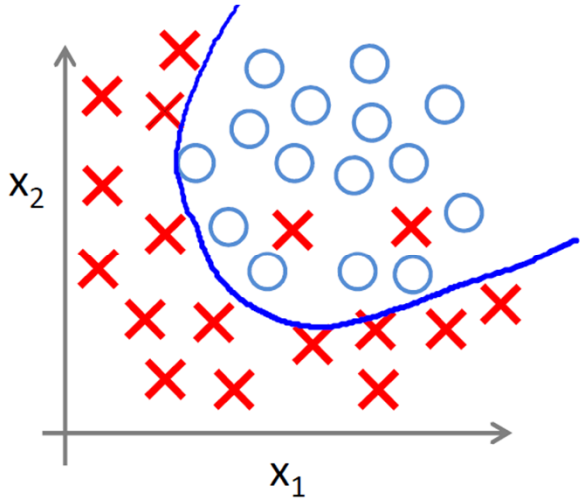


Overfitting problem

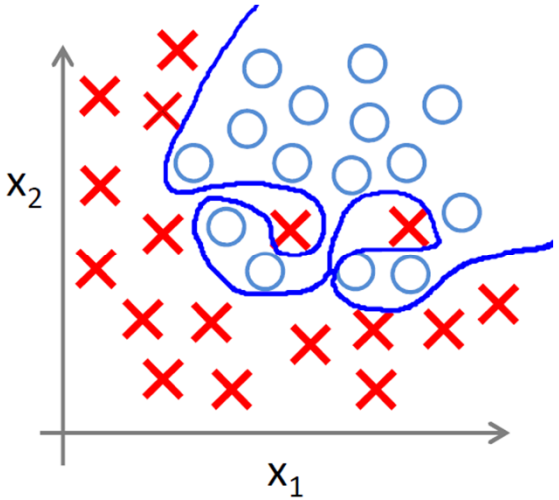
Overfitting: If we have too many features, the learned model may fit the training data very well but fail to generalize to new examples.



underfit- high bias model



ok model



overfit – high variance

Regularization

Regularization to prevent overfitting.

1 Ridge Regression

— Keep all the features, but reduces the magnitude of θ.

— Works well when each of the features contributes a bit to predict y.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

2 Lasso Regression

- May shrink some coefficients of θ to exactly zero.
- Serve as a feature selection tools (reduces the number of features).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\left|\theta_j\right|$$

Regularized Logistic Regression

Unregularized Logit cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Regularized Logit cost function (ridge regression)

λ is the regularization parameter (hyper-parameter) that needs to be selected

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

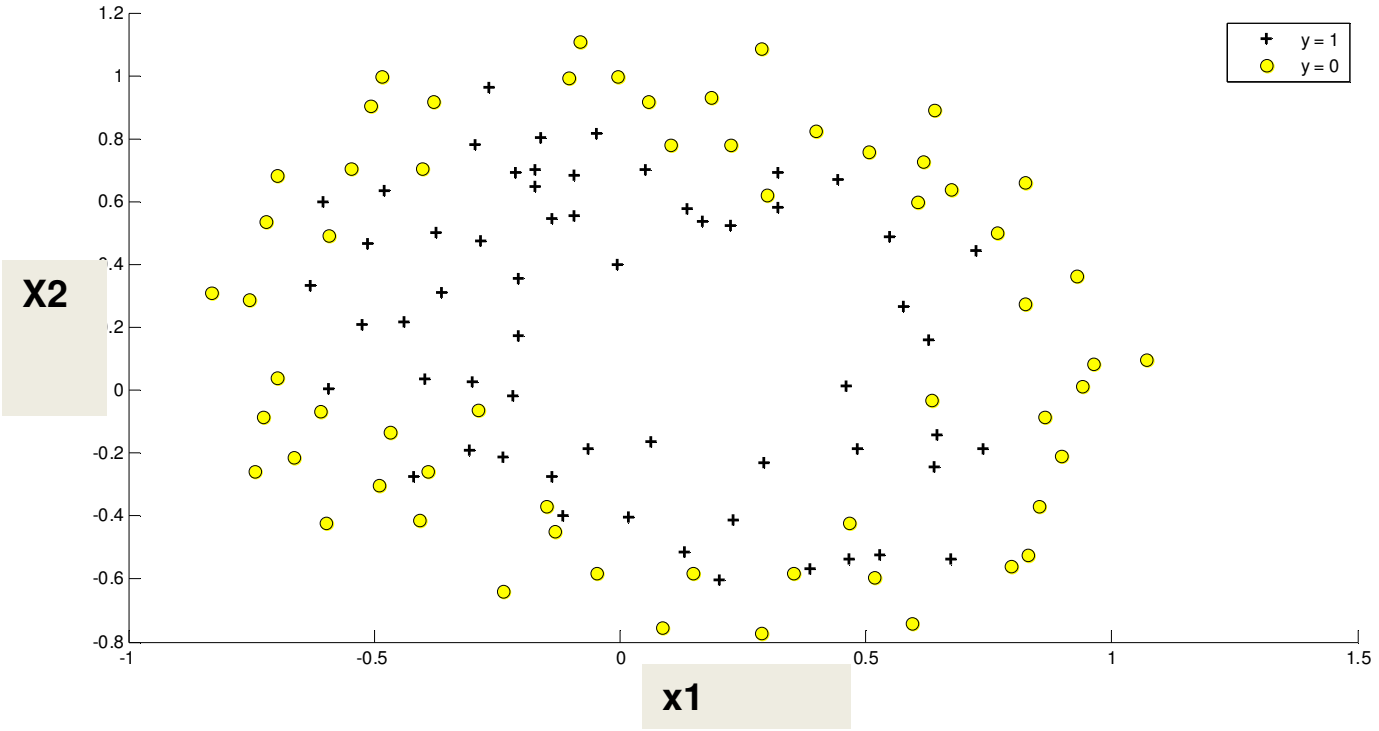
Small λ => lower bias, higher variance

High λ => higher bias, lower variance

Regularized Log Reg -example

2 features:
x1=mean uploaded packets/min
x2=mean downloaded packets/min

Label y: Network traffic OK (0) / Malware (1)



Regularized Log Reg -example

Dataset is not linearly separable => logistic regression will only be able to find a linear decision boundary. One way to fit the data better is to create more features. For example add polynomial terms of x1 and x2 .

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

NONLINEAR decision boundary $\Rightarrow$

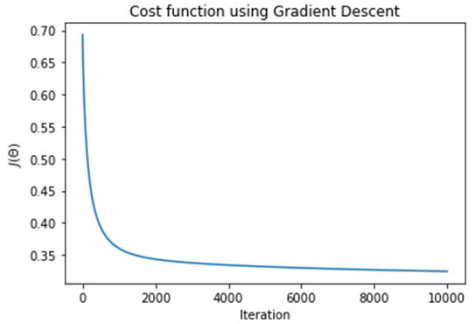
$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \ldots\ldots \theta_{28} x_2^6 = 0$$

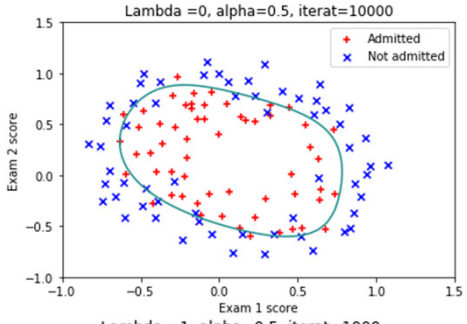
$$if \quad z > 0 \Rightarrow g(z) > 0.5 \Rightarrow \text{predict class} = 1$$

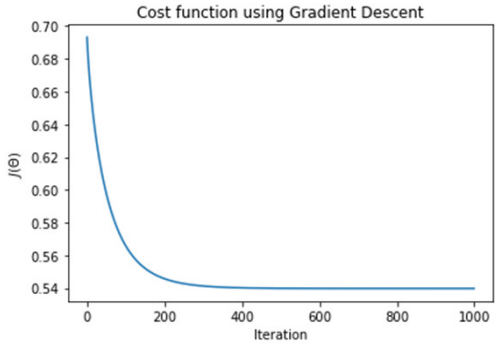
$$if \quad z < 0 \Rightarrow g(z) < 0.5 \Rightarrow \text{predict class} = 0$$

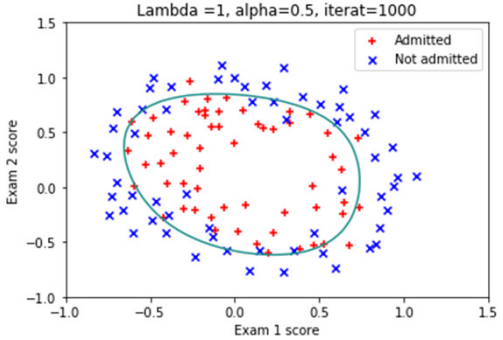
Regularized Log Reg -example

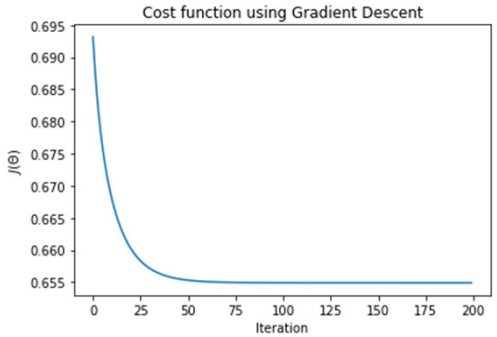
Accuracy on training data: :84.75% (λ=0) | 83.90 % (λ=1) | 71.2 % (λ=10))

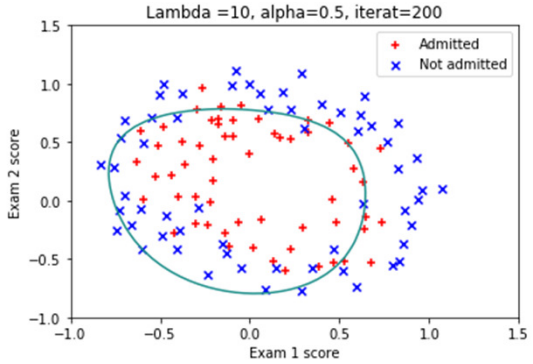












universidade de aveiro

Multiclass Classification

Ex. Network traffic OK (0) / Malware type A (1) / Malware type B (2) , etc.

One-versus-all strategy :

For K classes train K binary classifiers:

```
for c=1:K
      Make y_binary=1 (only for examples of class c)
            y_binary=0 (for examples of all other classes)
      theta=Train classifier with training data X and output y_binary.
      Save the learned parameters of all classifiers in one matrix
      where each raw is the learned paramenters of one classifier:
     theta_all(c,:)=theta
end
```

New example: winner-takes-all strategy, the binary classifier with the highest output score assigns the class.

ML: Theory vs Implementation

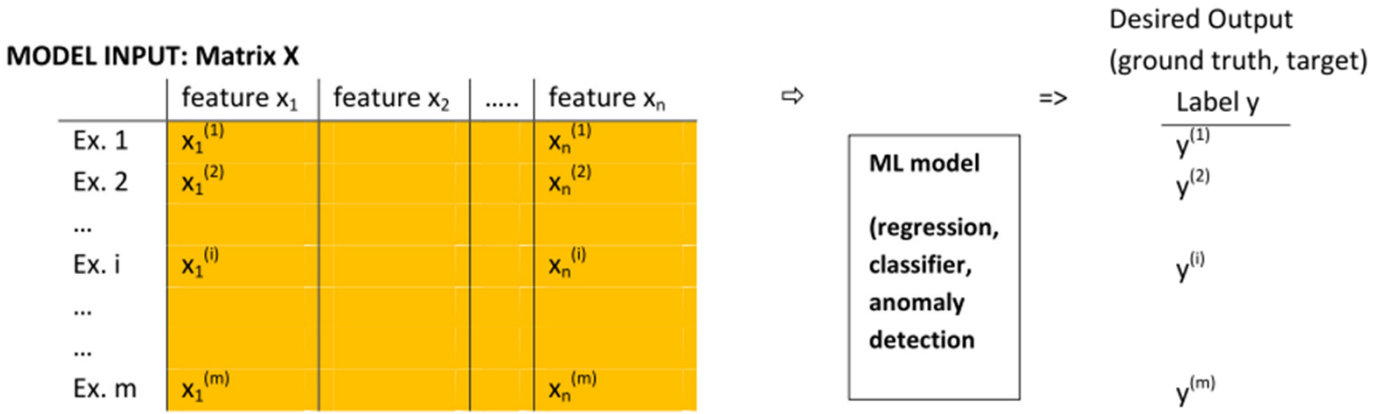
Theory: x – vector of features; θ – vector of model parameters; ML (math) models

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}; \quad \vec{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... \theta_n x_n = \vec{\theta}^T \vec{x}$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Implementation: Given labelled data of m examples, n features

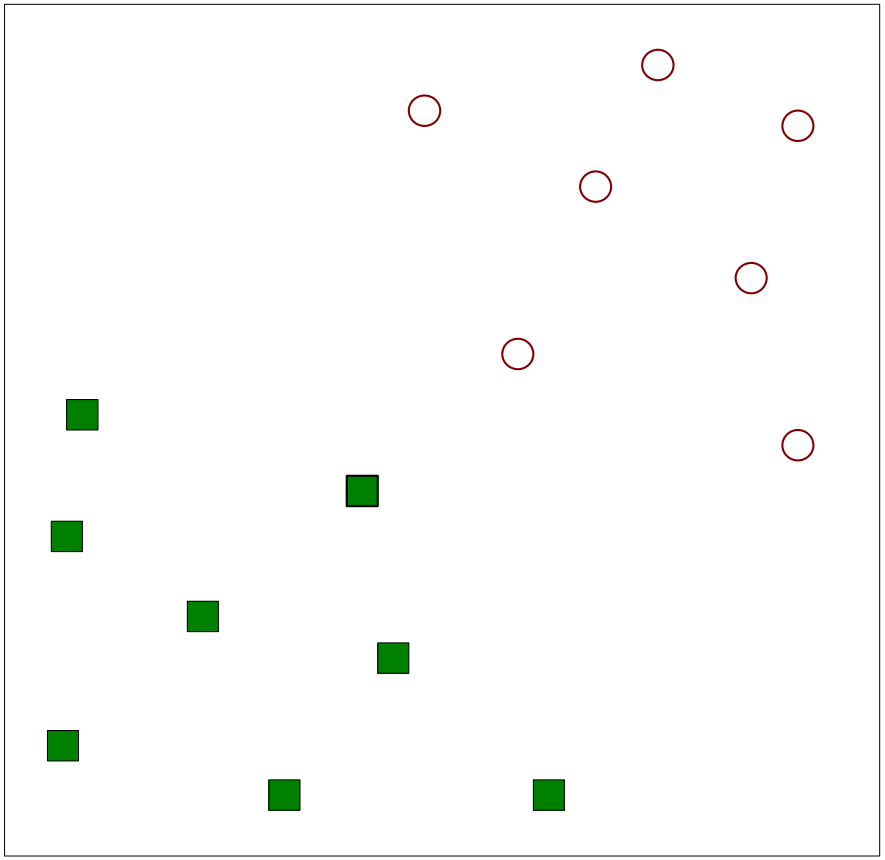


Classification –

SUPPORT VECTOR MACHINES (SVM)

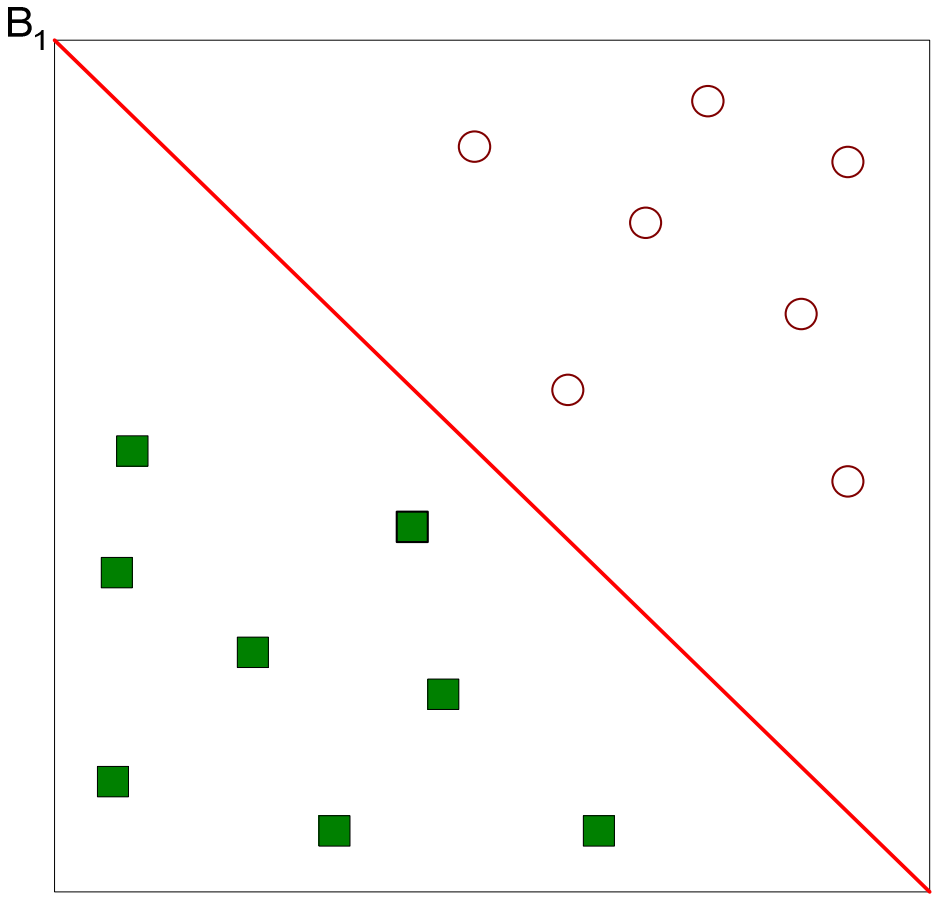
Proposed by Vladimir N. Vapnik and Alexey Chervonenkis, 1963

Linearly separable classes



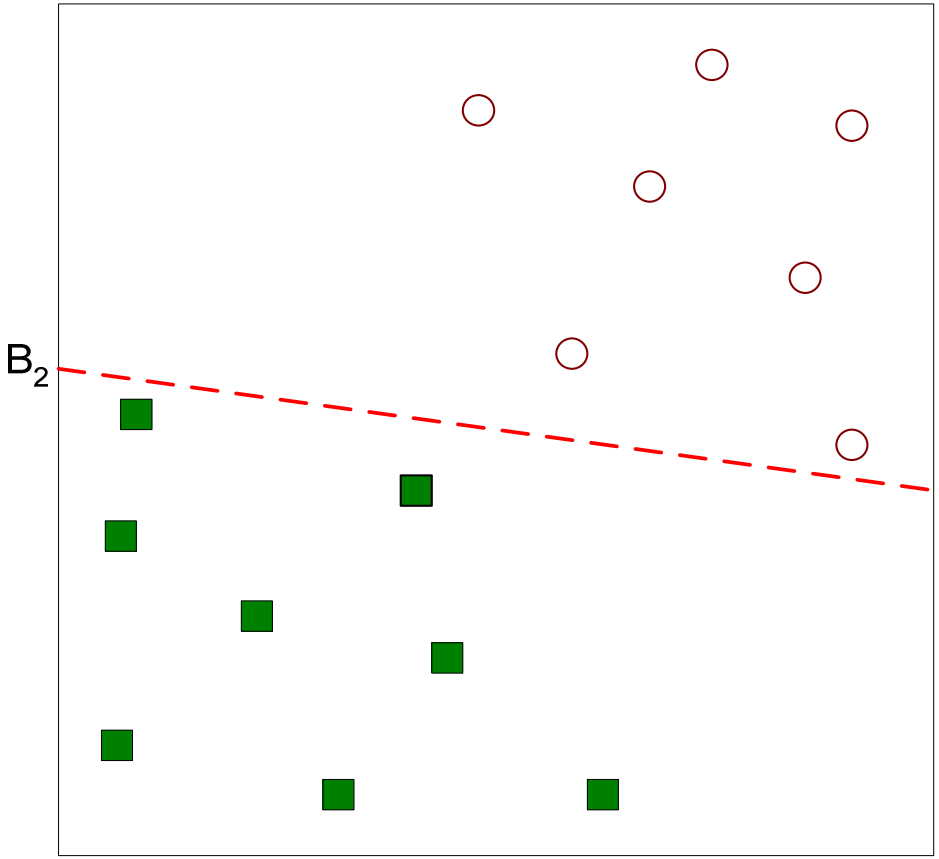
Find a decision boundary to separate data

Linearly separable classes



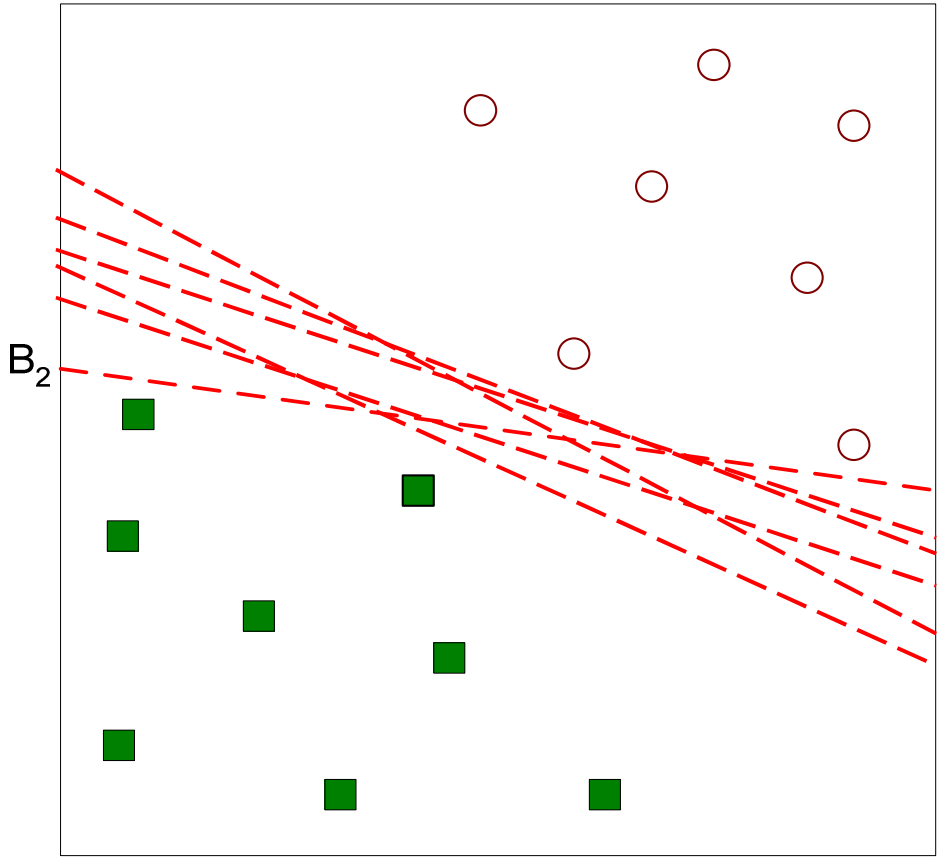
One Possible Solution

Linearly separable classes



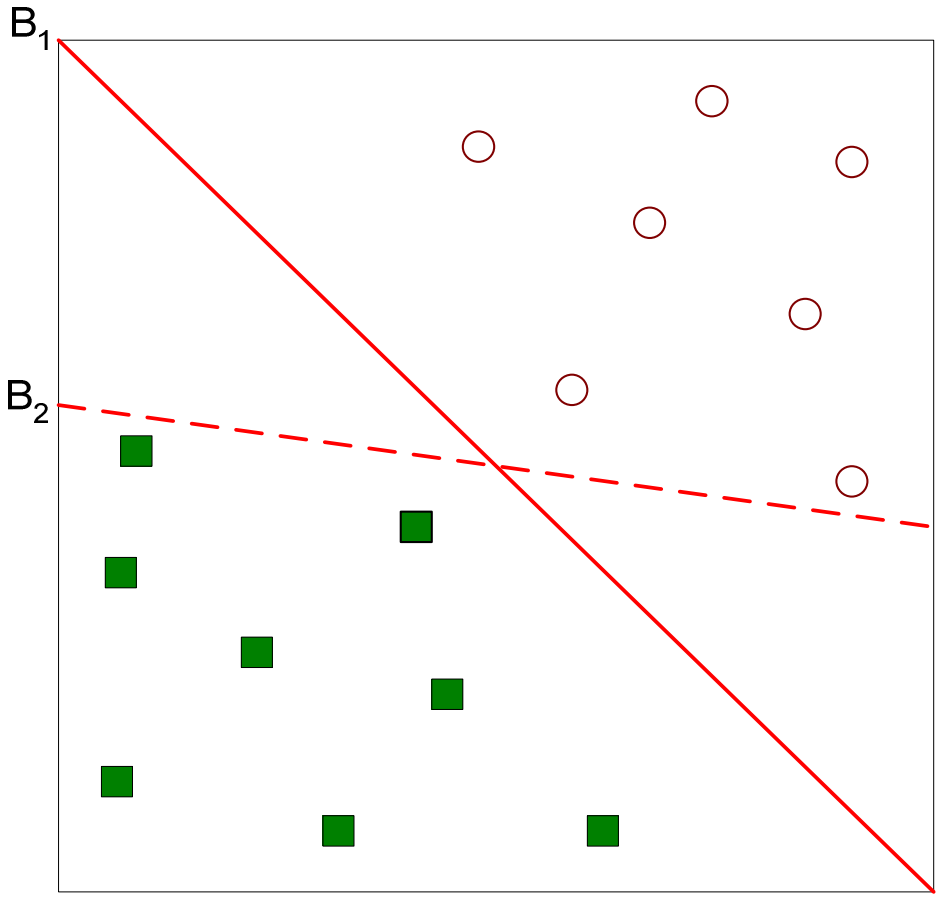
Another possible solution

Linearly separable classes



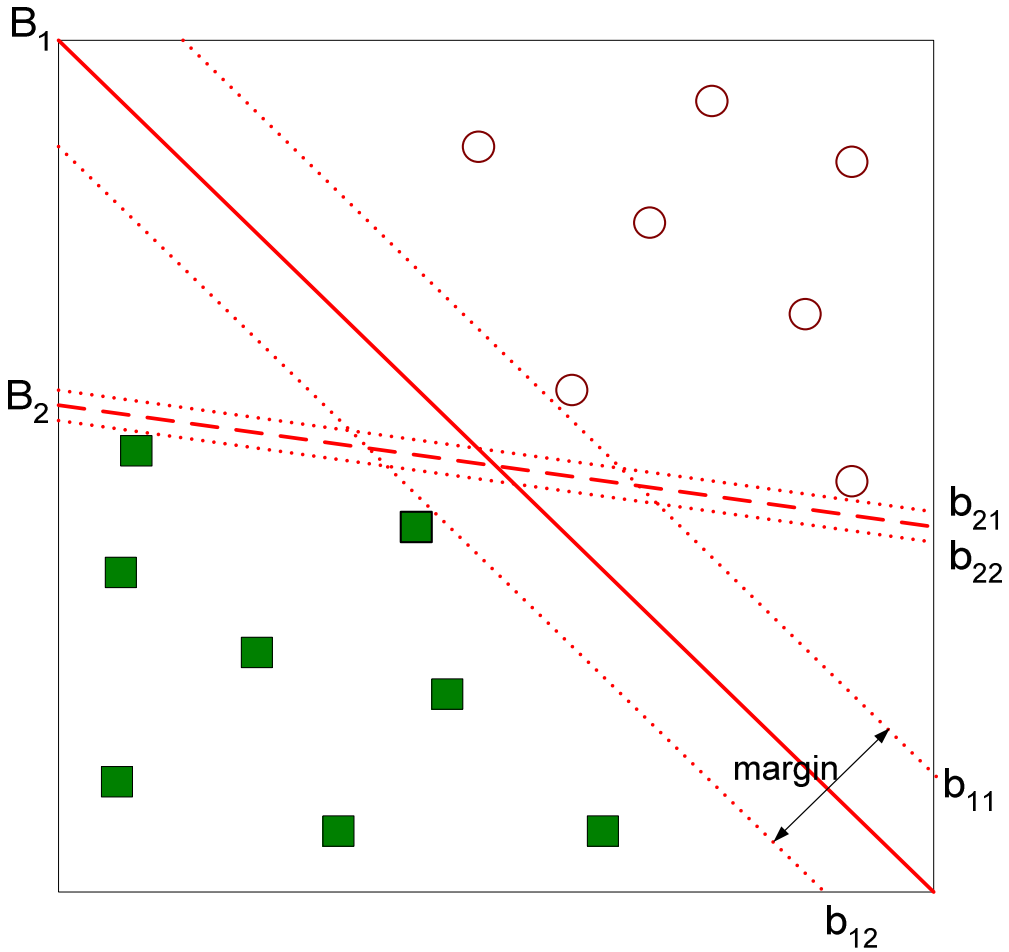
Many possible solutions

Linearly separable classes



Which one is better? B1 or B2?

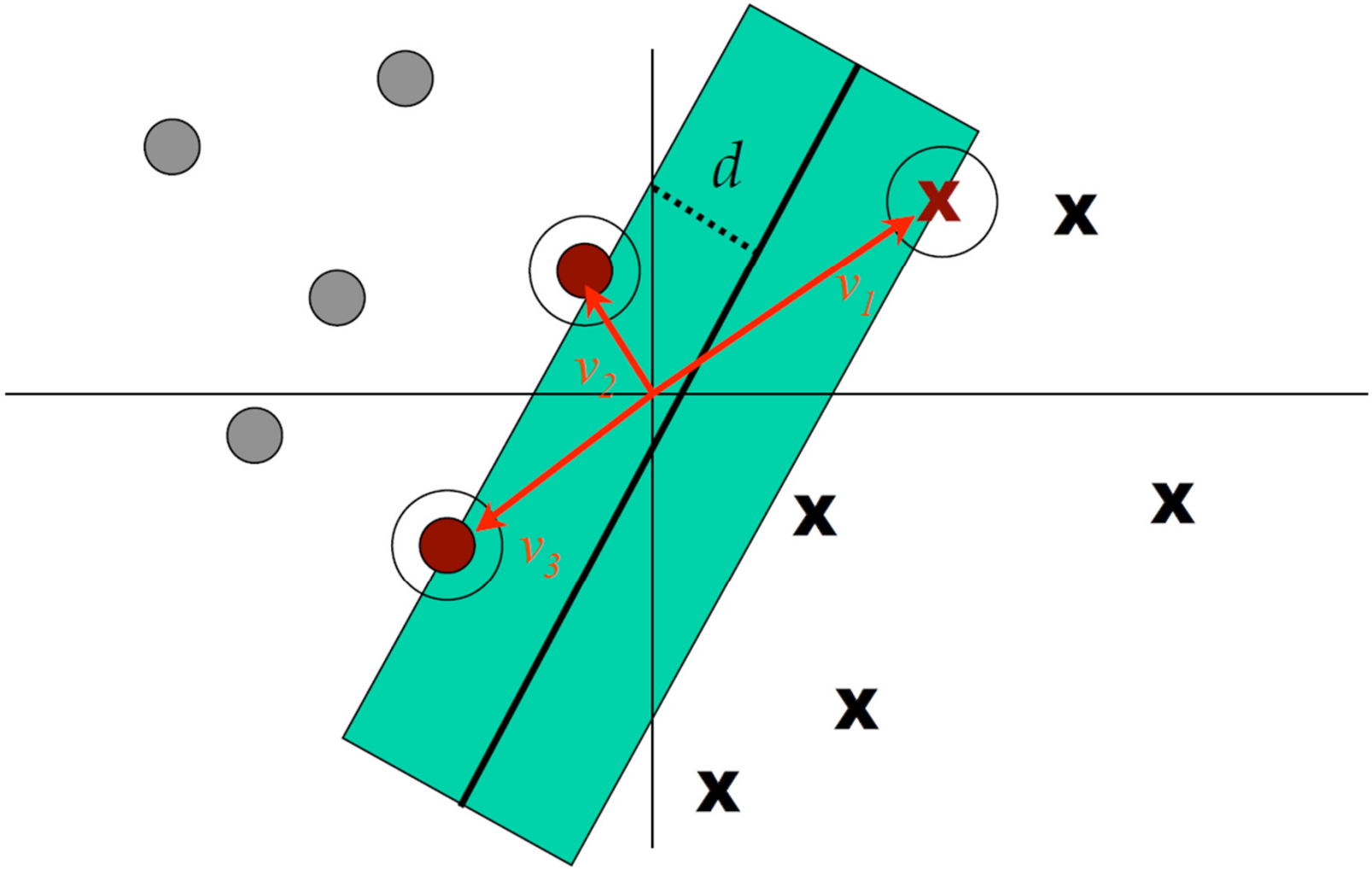
SVM - Large margin classifier



Find a boundary that maximizes the margin => B1 is better than B2

SUPPORT VECTORS (v1,v2,v3)

Only the closest points (support vectors) from each class are used to decide which is the optimum (the largest) margin between the classes.

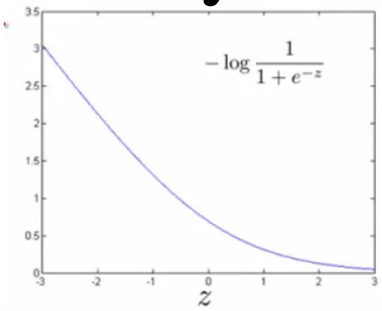


SVM cost function

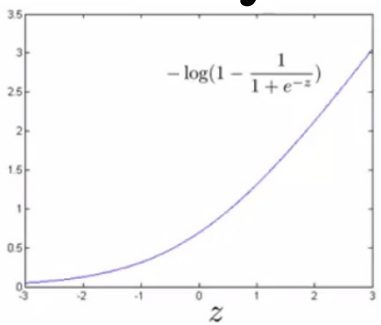
Regularized LogReg cost function:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

for y=1

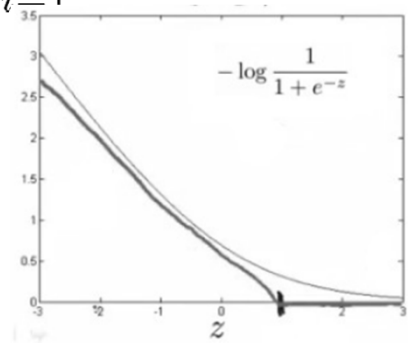


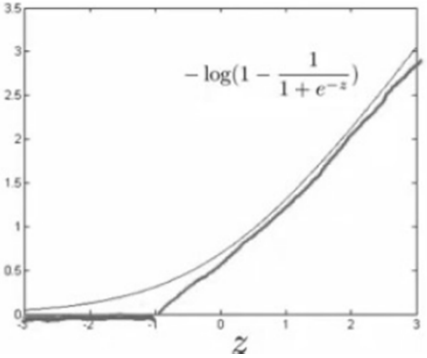
for y=0



Regularized SVM cost function (Modification of Logit cost function.
cost0 & ***cost1*** are assimptotic safety margins with computational advantages)

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$





$$z = \theta^T x$$

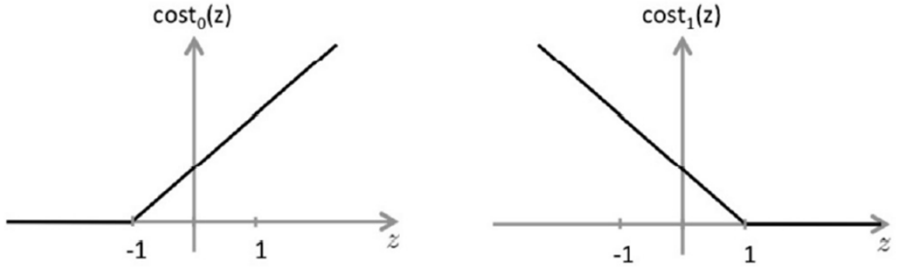
SVM cost function

Regularized LogReg cost function:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Regularized SVM cost function

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

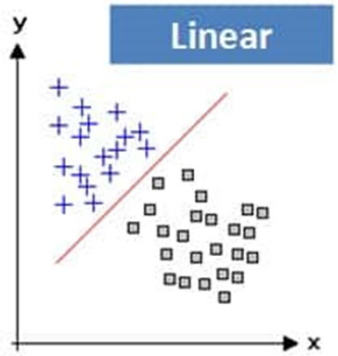


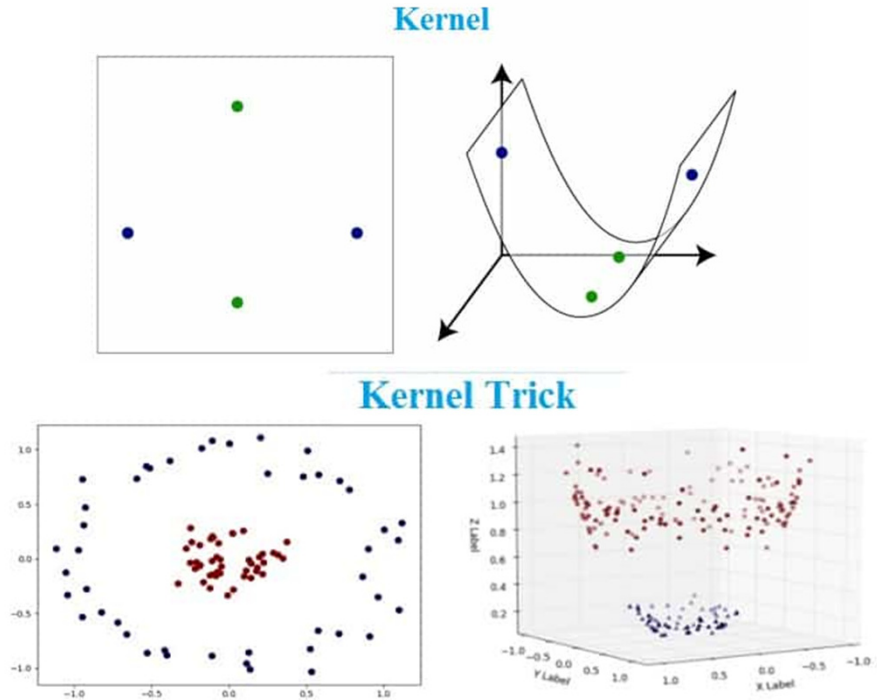
$$z = \theta^T x$$

Different way of parameterization: C is equivalent to $1/\lambda$.

C >0 - parameter that controls the penalty for misclassified training examples.
Increase C more importance to training data fitting.
Decrease C – more importance to generalization properties (combat overfitting).

Nonlinearly separable data – kernel SVM





Kernel: function which maps a lower-dimensional data into higher dimensional data.

Tipical Kernels:

- Polynomial Kernel - adding extra polynomial terms
- Gaussian Radial Basis Function (RBF) kernel – the most used kernel
- Laplace RBF kernel
- Hyperbolic tangent kernel
- Sigmoid kernel, etc.

Nonlinear SVM – Gaussian RBF Kernel

$$k(x_i, x_j) = e^{\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right)}, \quad \gamma > 0, \gamma = 1/2\sigma^2, \quad \sigma - \text{variance}$$

The RBF kernel is a metric of similarity between examples, $x^{(i)}$ and $x^{(j)}$
Substitute the original features with similarity features (kernels).

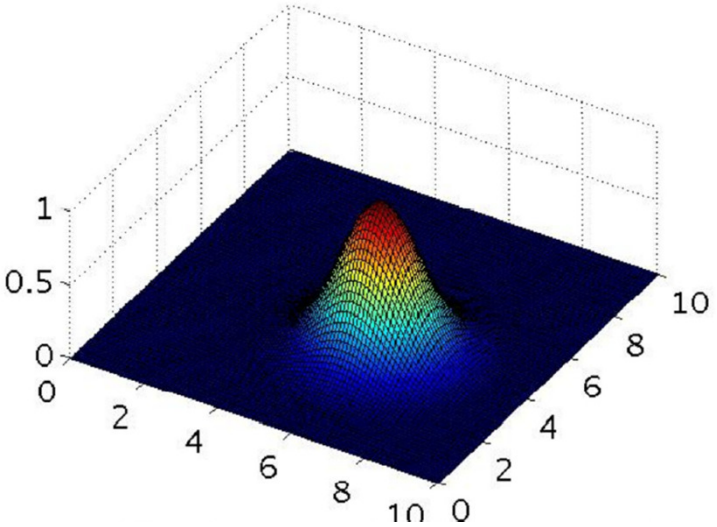
Note: the original (n+1 dimensional) feature vector is substituted by the new (m+1 dimensional) similarity feature vector.

m –number of examples, **m>>n !!!**

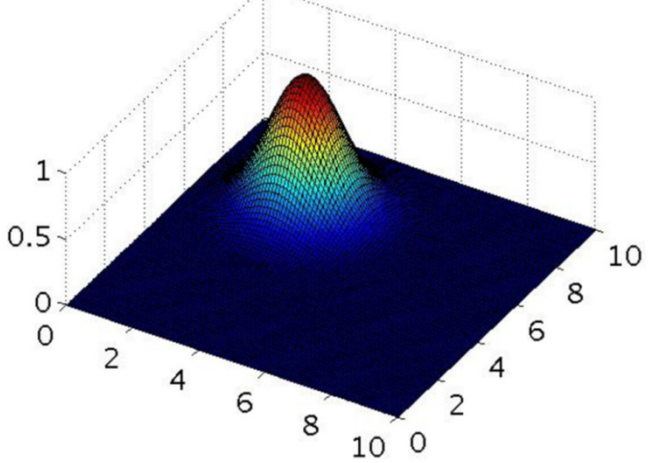
Gaussian RBF Kernel – Parameter σ

$$k(x_i, x_j) = e^{-\gamma \left\| x_i - x_j \right\|^2}, \quad \gamma = \frac{1}{2\sigma^2} > 0$$

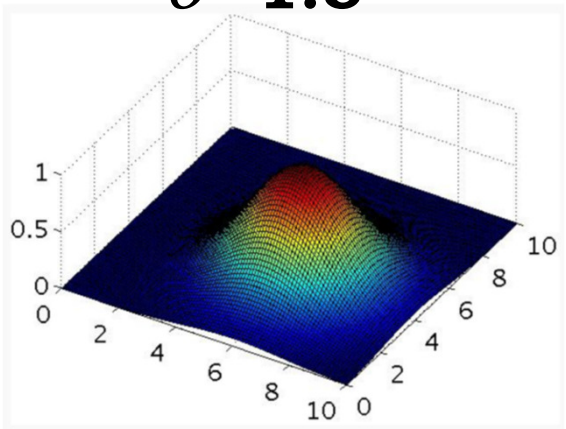
σ=1



σ=0.5



σ=1.5



σ determines how fast the similarity metric decreases to 0 as the examples go away of each other.

Large σ: kernels vary more smoothly (combat overfitting)

Small σ: kernels vary less smoothly (more importance to training data fitting)

SVM parameters

How to **choose hyper-parameter C:**

Large C: lower bias, high variance (equivalent to small regular. param. λ)

Small C: higher bias, lower variance (equivalent to large regular. param. λ)

How to **choose hyper-parameter** σ**:**

Large σ**:** features vary more smoothly. Higher bias, lower variance

Small σ**:** features vary less smoothly. Lower bias, higher variance

SVM implementation

Use SVM software packages to solve SVM optimization !!!

In Python, use Scikit-learn (sklearn) machine learning library and

Import SVC (Support Vector Classification):

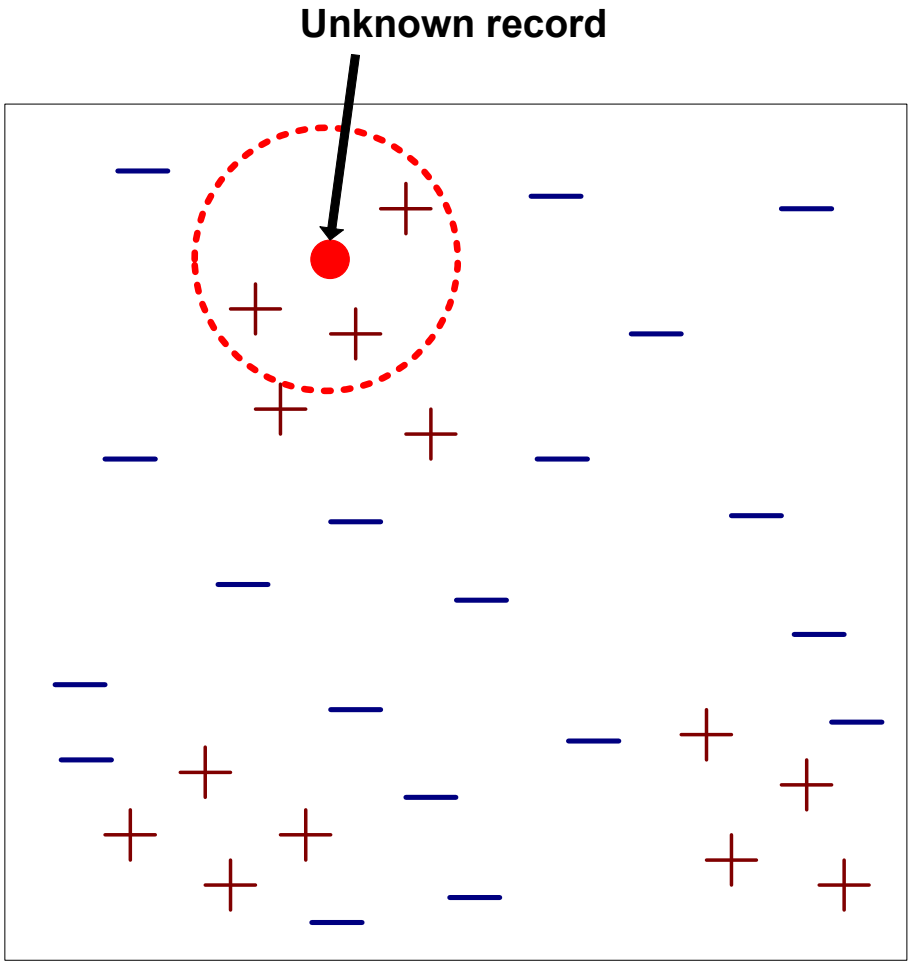
from sklearn.svm import SVC
classifier = SVC(kernel="rbf",gamma =?)

"rbf“ (Radial Basis Function) corresponds to the Gaussian kernel.
gamma = 1/σ.

Classification –

K- Nearest-Neighbor (k-NN)

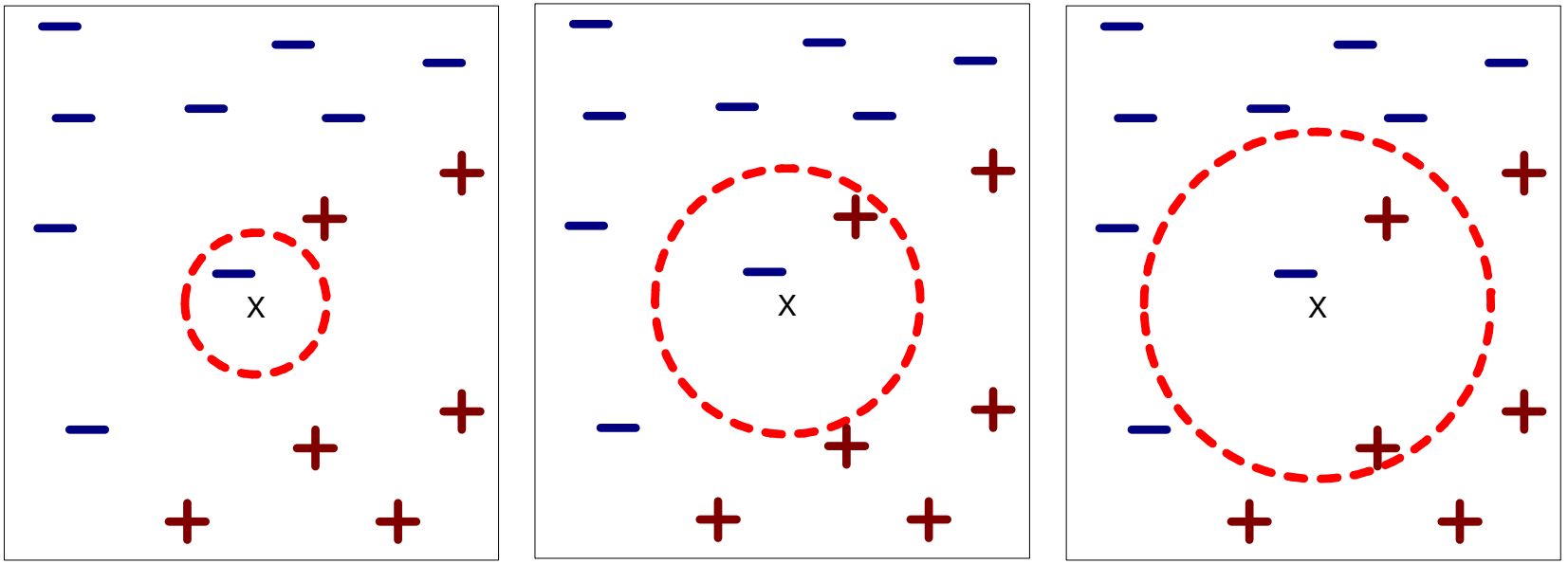
K- Nearest-Neighbor (k-NN) Classifier



- KNN requires:
  - Set of labeled records.
  - Measure to compute distance (similarity) between records.
  - *K* is the number of nearest neighbors (the closest points).

- To classify a new (unlabeled) record:
  - Compute its distance to all labeled records.
  - Identify *k* nearest neighbors.
  - The class label of the new record is the label of the majority of the nearest neighbors.

K-NN- choice of k



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K- Nearest Neighbors of the new point x are the points that have the smallest distance to x

Lab work - Spam Classification

- Labelled data set : SpamAssassin Public Corpus

- Convert the email into a binary feature vector:

- Clean (remove slash , dots, coms)
- Tokenize (parse) into words
- Count the word frequency
- Create dictionary with most frequent words (e.g. 10000 to 50000 words) -

Feature space.

- Substitute the words with the dictionary indices.
- Extract binary features from emails - binary (sparse) feature for an email corresponds to whether the i-th word in the dictionary occurs in the email.

- **Apply classifier** (e.g. SVM)

Dictionary => Email with dictionary indices => binary features

```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```

```
86 916 794 1077 883
370 1699 790 1822
1831 883 431 1171
794 1002 1893 1364
592 1676 238 162 89
688 945 1663 1120
1062 1699 375 1162
479 1893 1510 799
1182 1237 810 1895
1440 1547 181 1699
1758 1896 688 1676
992 961 1477 71 530
1699 531
```

$$x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n$$

universidade de aveiro