ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Machine Learning

LECTURE 5:

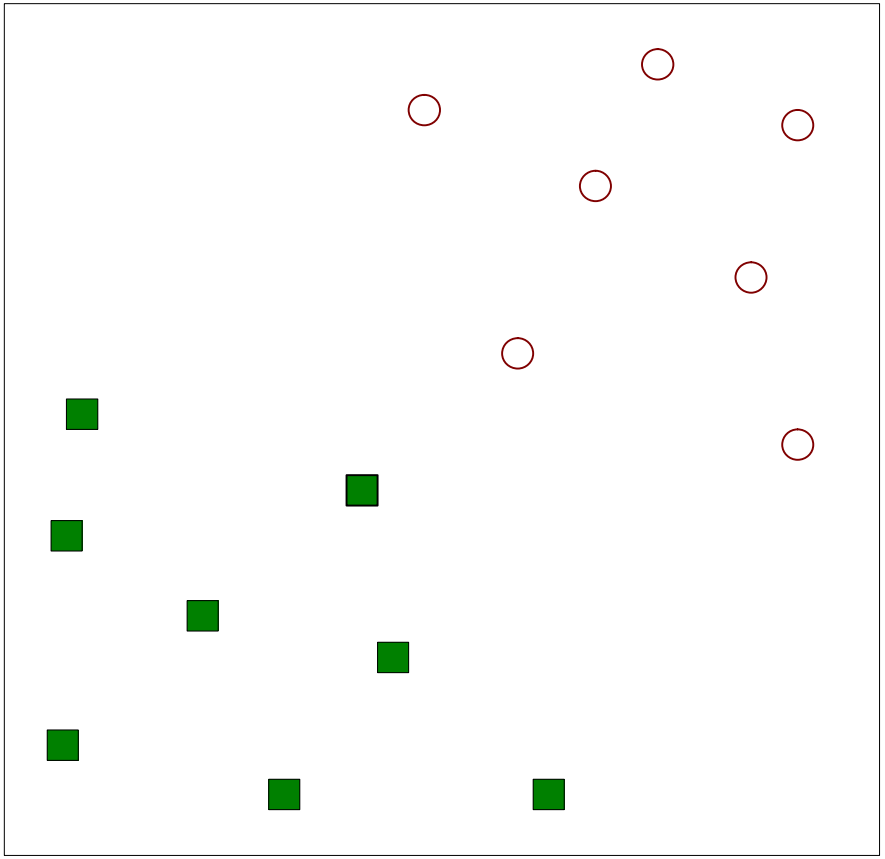
SUPPORT VECTOR MACHINE (SVM)

Petia Georgieva
(petia@ua.pt)

Lecture outline

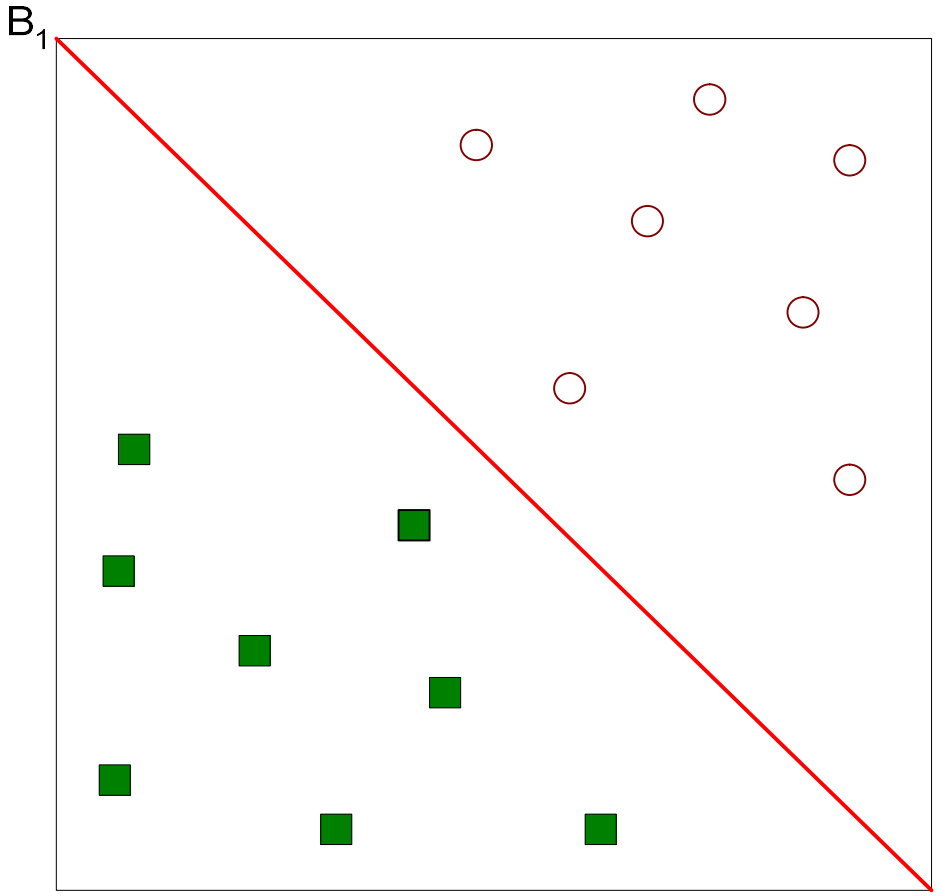
1. Linear Support Vector Machine (SVM)
2. Nonlinear SVM - Gaussian RBF Kernel
3. Performance evaluation – confusion matrix
4. Training, validation, testing – three way data split

Linearly separable classes



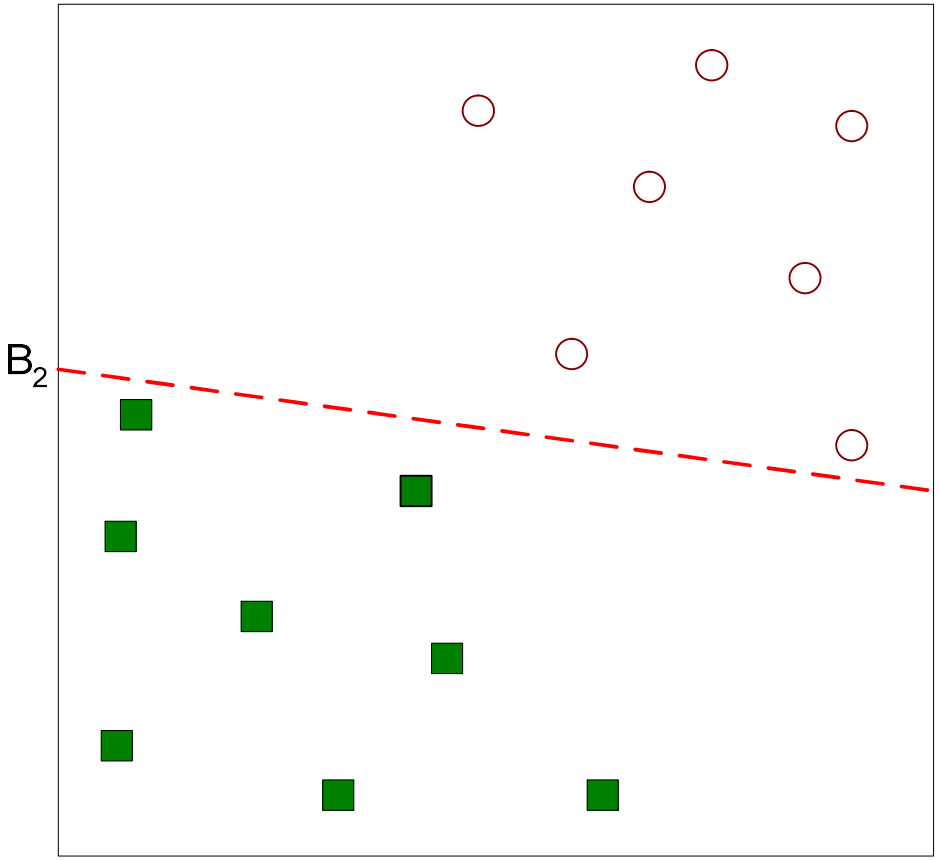
Find a decision boundary to separate data

Linearly separable classes



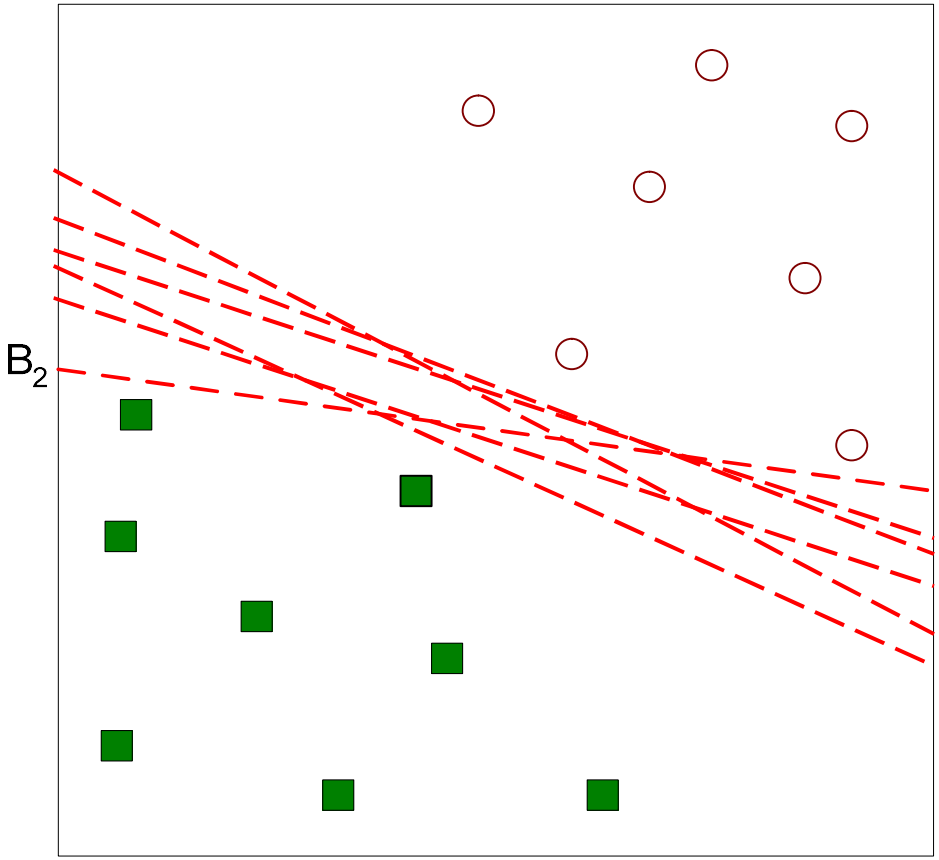
One Possible Solution

Linearly separable classes



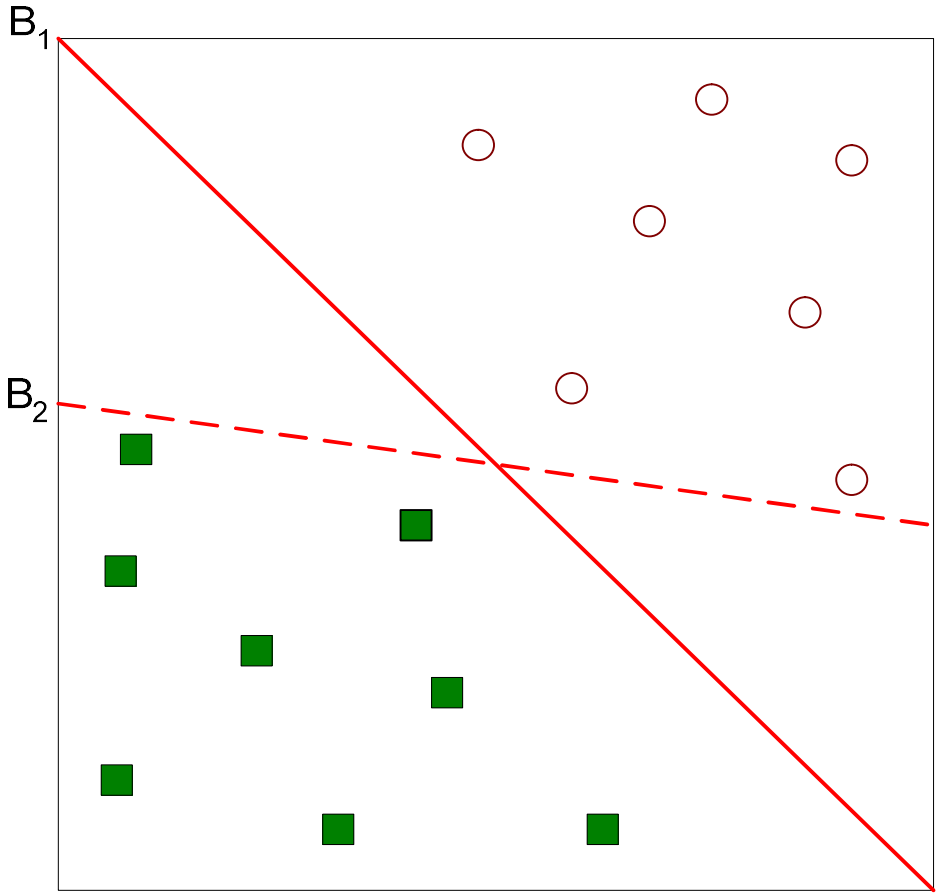
Another possible solution

Linearly separable classes



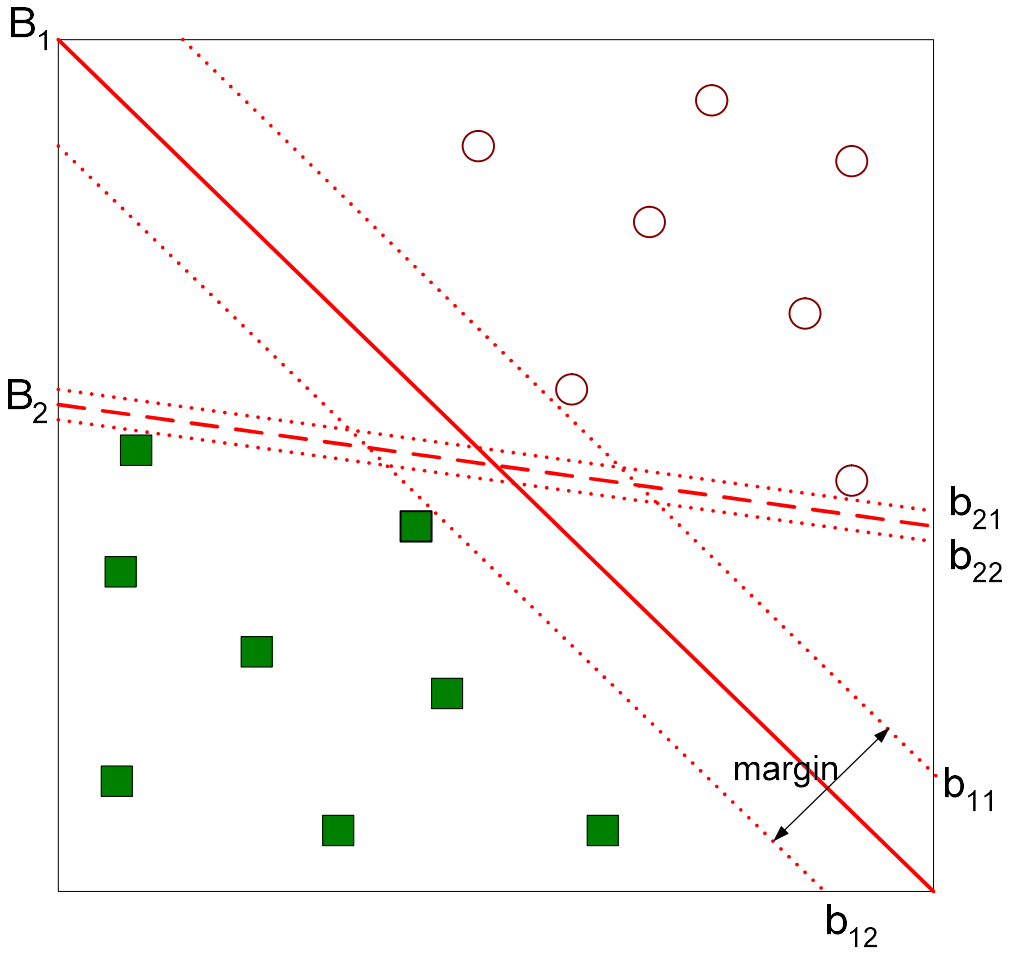
Many possible solutions

Linearly separable classes



Which one is better? B1 or B2?

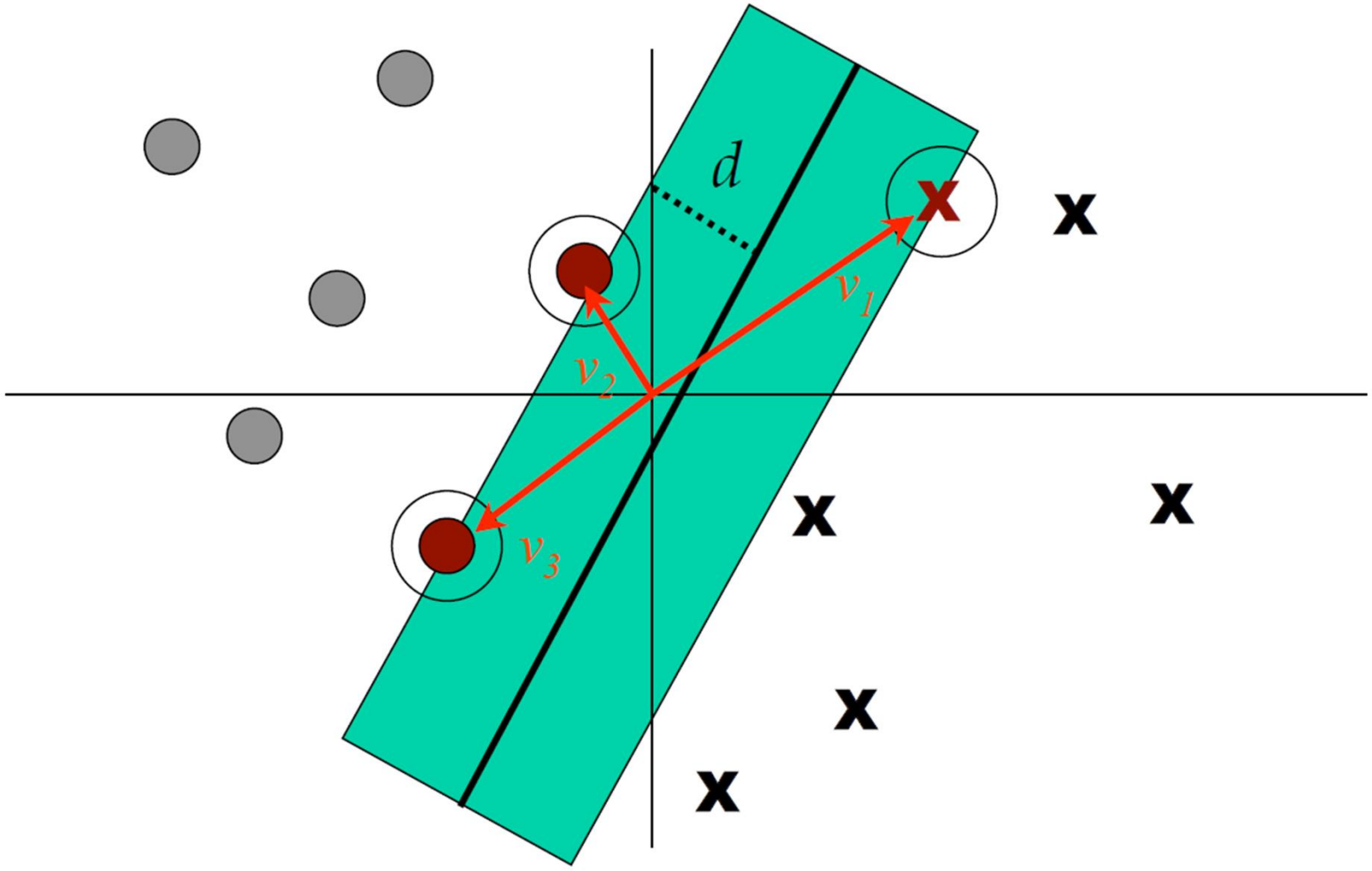
SVM - Large margin classifier



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.

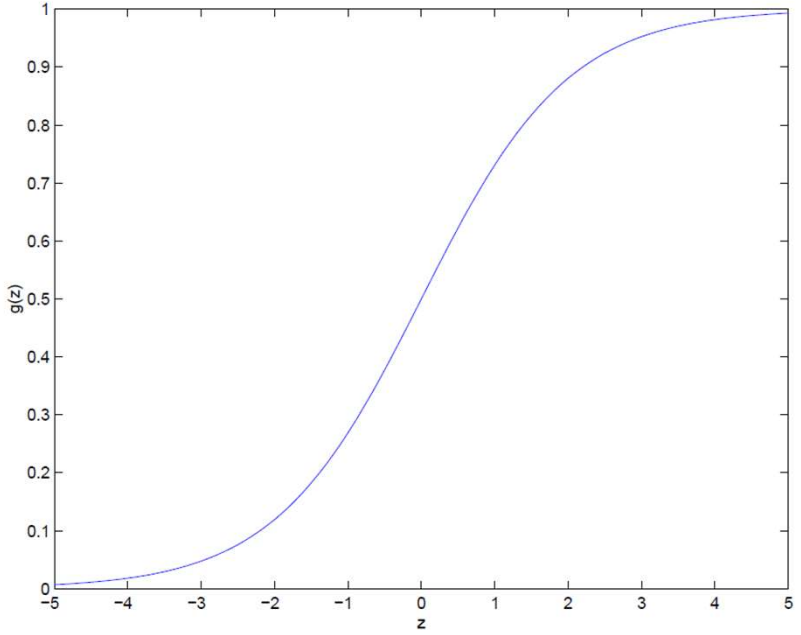


Logistic Regression (LogReg) -revised

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \qquad \theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

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

Logistic (sigmoid) function



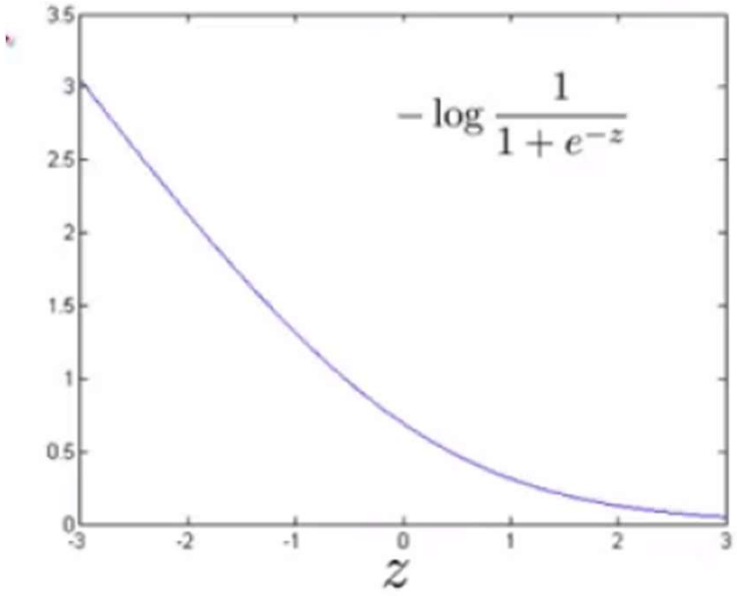
if $y = 1$, we want $h_\theta(x) \approx 1, \quad \theta^T x >> 0$

if $y = 0$, we want $h_\theta(x) \approx 0, \quad \theta^T x << 0$

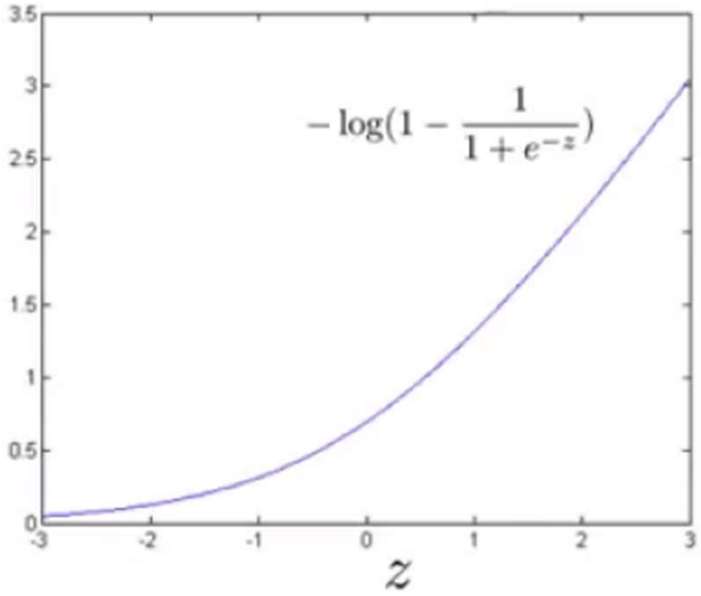
LogReg cost function -revised

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

for y=1



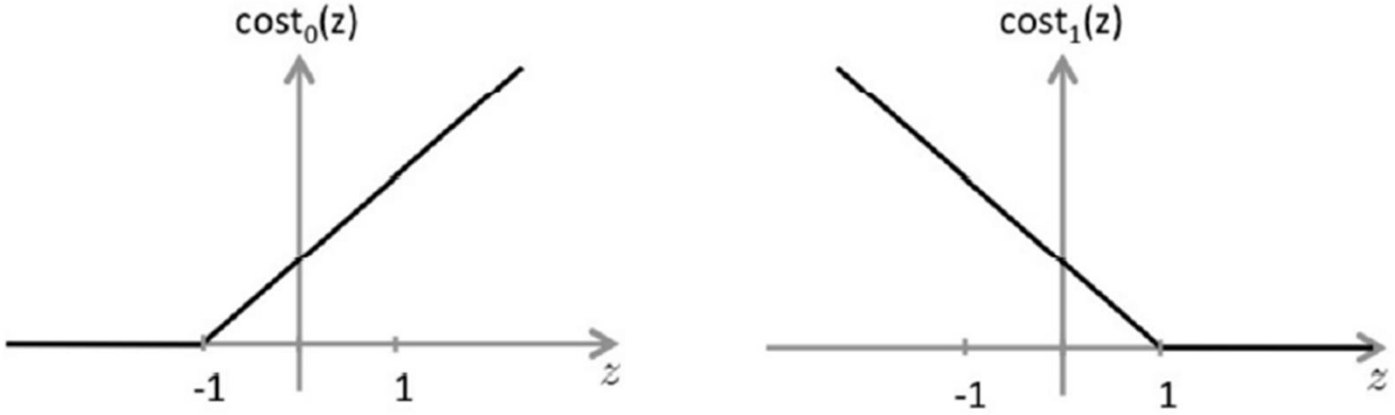
for y=0



SVM cost function

Modification of LogReg cost function:
cost0 and ***cost1*** are approxiamate assimptotic margins, add safety margin and have computational advantages.

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



if $y = 1$, we want $\theta^T x \geq 1$ (not just > 0)

if $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

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 (different parametrization)

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

Different way of parameterization: instead of λ now we have C.
The two optimization methods will give the same optimal value of θ if C =>1/ λ

C >0 - parameter that controls the penalty for misclassified training examples.
Large C (C>>1) tells SVM to try to classify all examples correctly, e.g. the first term will tend to 0.

SVM optimization objective

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

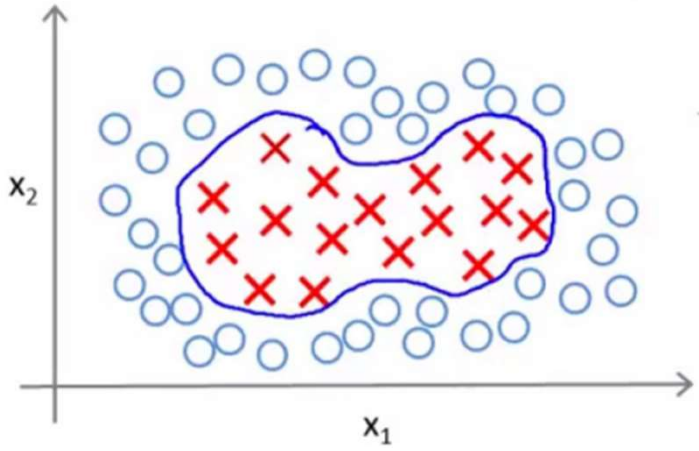
If C >> 1 the optimization is reduced to:

$$\min_{\theta} \sum_{j=1}^{n} \theta_j^2, \quad \text{such that}$$

$$\theta^T x^{(i)} \geq 1, \quad \text{if } y = 1$$

$$\theta^T x^{(i)} \leq -1 \quad \text{if } y = 0$$

Nonlinearly separable data – kernel SVM



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 Kernel

$$k(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|^2\right), \quad \gamma > 0, \gamma = 1/2\sigma^2$$

The kernel functions define metrics of similarity between examples.
Substitute the original features with new (similarity) features (the kernels).

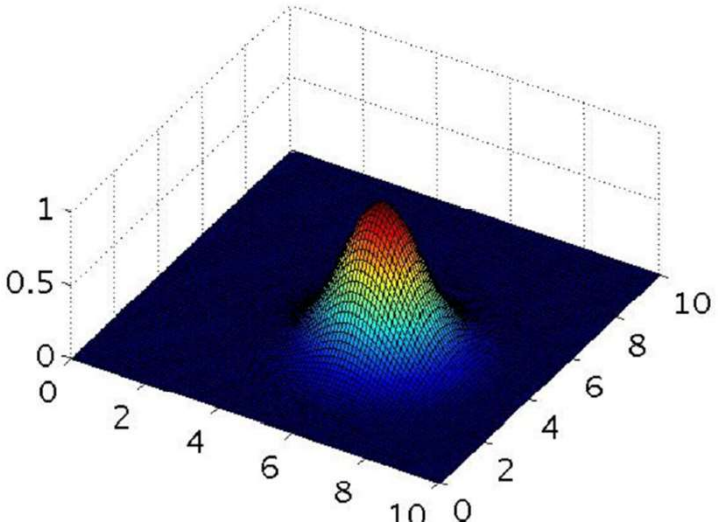
Note: the original (n+1 dimensional) feature vector is substituted by the new (m+1 dimensional) feature vector.

m –number of examples, **m>>n !!!**

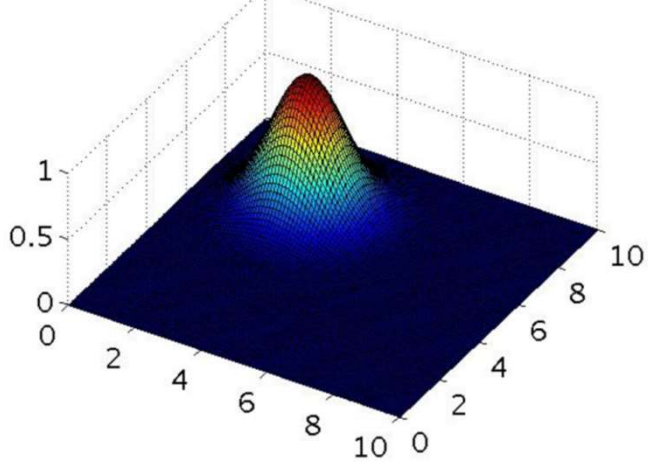
Gaussian RBF Kernel – Parameter σ

$$k(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|^2\right), \quad \gamma > 0, \gamma = 1/2\sigma^2$$

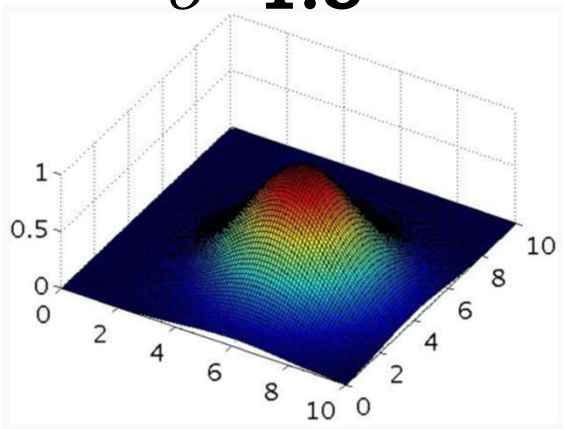
σ=1



σ=0.5



σ=1.5



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

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.

Logistic Reg versus SVM

n = number of features, m=number of examples

- If n is large (relative to m) (e.g. n=10000; m=10-1000) =>

use logistic regression or SVM without kernel (“linear kernel”)

- If n is small, m is intermediate (n=1-1000; m=10-10000) =>

Use SVM with Gaussian kernel

- If n is small, m is large (n=1-1000; m=50000)

Create more features, then use logistic regression or SVM without a kernel.

- Neural Networks likely to work well for most of these setting, but may be slower to train.

Performance Evaluation – Confusion Matrix

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Performance metric - Accuracy

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	(TP)	(FN)
	Class=No	(FP)	(TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy - fraction of examples correctly classified.

1-Accuracy: Error rate (misclassification rate)

Limitation of Accuracy

- Consider binary classification (**Unbalanced data set**)
  - Class 0 has 9990 examples
  - Class 1 has 10 examples

- If model classify all examples as class 0, accuracy is 9990/10000 = 99.9 %

- Accuracy is misleading because model does not classify correctly any example of class 1 => Need to find a way to balance the data set !!!

Other performance metrics

Sensitivity (recall) – true positive rate, of all positive examples the fraction of correctly classified

$$\text{Recall (r)} = \frac{TP}{TP + FN}$$

Specificity - true negative rate, of all negative examples the fraction of correctly classified

$$\text{Specificity(s)} = \frac{TN}{TN + FP}$$

Precision - the fraction of correctly classified positive samples from all classified as positive

$$\text{Precision (p)} = \frac{TP}{TP + FP}$$

F1 Score - weighted average of Precision and Recall

F1=2*(Recall * Precision) / (Recall + Precision)

Performance metrics – example

	predicted	
	Positive	Negative
Positive	500	100
Negative	500	10000

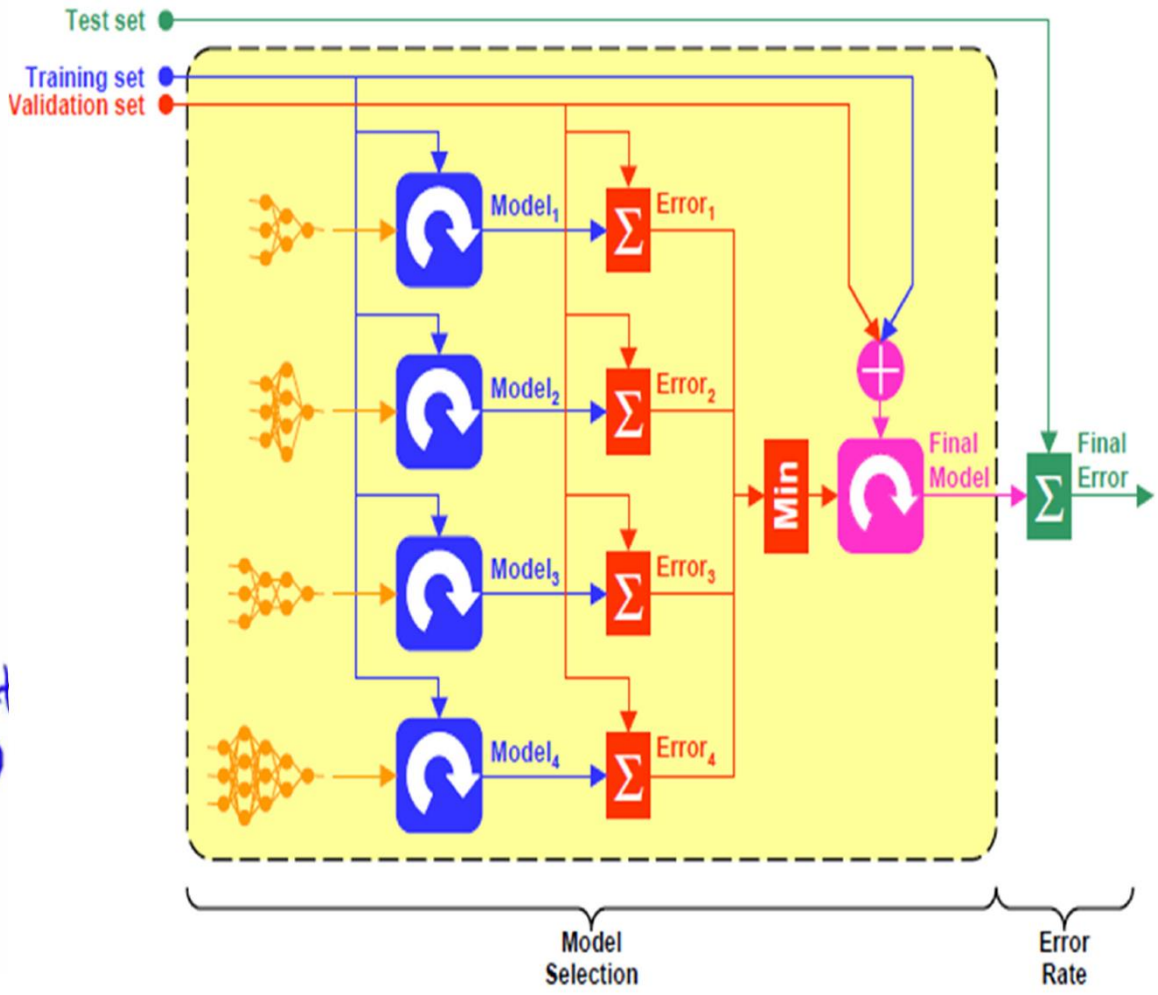
- Accuracy $\frac{500+10000}{500+500+100+10000} = 0.95$
- Precision $\frac{500}{500+500} = 0.5$
- Recall $\frac{500}{500+100} = 0.83$
- Specificity $\frac{10000}{10000+500} = 0.95$

- Positive class is predicted poorly
- Accuracy is not a reliable measure for un-balanced datasets
- If # of examples of one class is much lower than # of examples of the other class => Precision and Recall are better measures.

Training/Validation/Test subsets

Dataset:

Size	Price	
2104	400	
1600	330	
2400	369	60% Training set
1416	232	
3000	540	
1985	300	
1534	315	20% Cross validation set (CV)
1427	199	
1380	212	20% test set
1494	243	



The most important and credible is the final error (obtained with the test set, not used for training or validation of the model)

3 – way data split

Divide data into **training, validation and test** subsets

Stage 1 Train different models: (e.g **LR, ANN, SVM**, etc.) or change the hyper parameters (e.g. **# of hidden layer units, # of layers, C,** σ **,** λ , etc.)

Repeat: j=1: number of models

1. Train Model_J with the training set => get train error **E_train**

2. Use Model_J to predict validation set => get validation error **E_val**

3. Select the best model: the one that gives minimum **E_val**

Stage 2 Final tuning: train the best model again using data from both training and validation set (starting from the optimal model parameters computed at the previous training stage).

Stage 3 Test the final model: to predict the test set => get test error **E_test**

E_test is what you have to give to your clients !!!