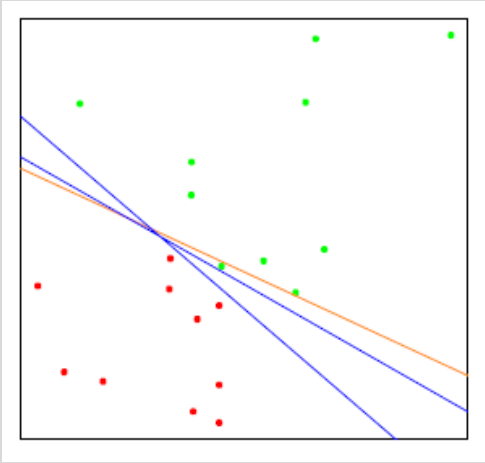


# Parallel & Scalable Machine Learning

Introduction to Machine Learning Algorithms

**Dr. – Ing. Morris Riedel**
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland
Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 7

# Support Vector Machines and Kernel Methods

January 16th, 2018
JSC, Germany

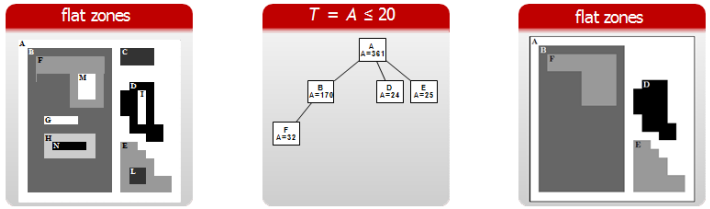
UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

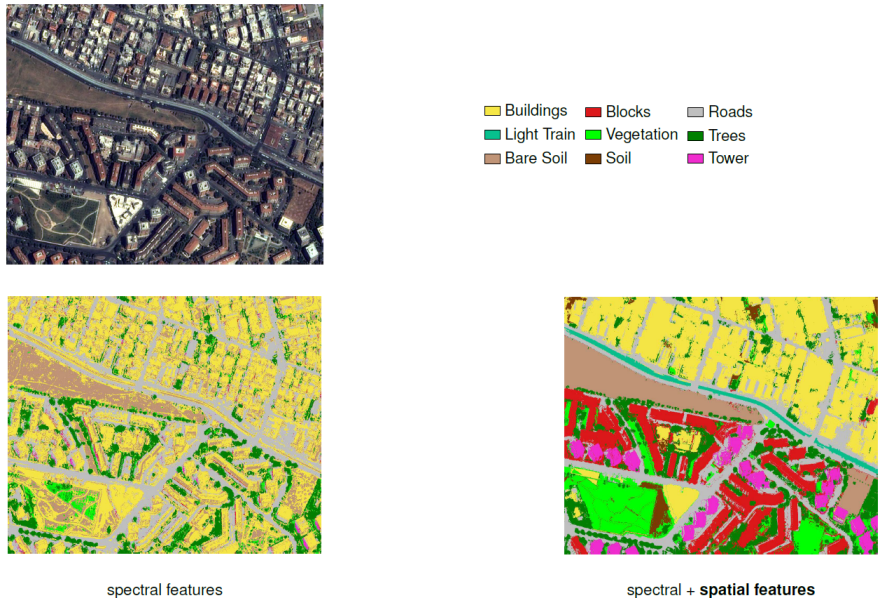
JÜLICH FORSCHUNGSZENTRUM

PRACE PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE
TRAINING PORTAL

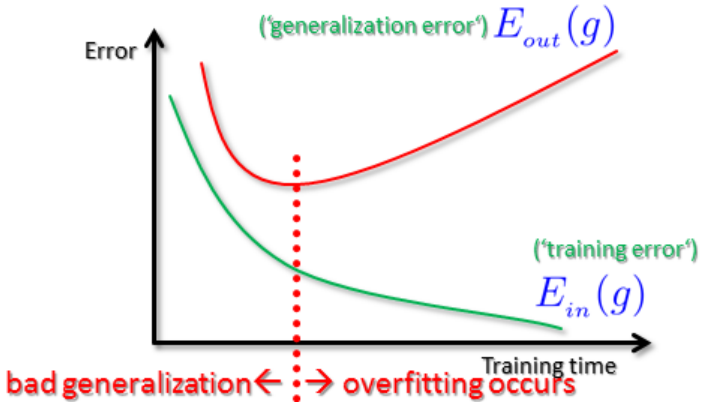
# Review of Lecture 6

- Remote Sensing Data
  - Pixel-wise spectral-spatial classifiers
  - Feature enhancements
  - Example of Self-Dual Attribute Profile (SDAP) technique




- Classification Challenges
  - Scalability, high dimensionality, etc.
  - Non-linearly seperable data
  - Overfitting as key problem in training machine learning models



# Outline

# Outline of the Course

1. Introduction to Machine Learning Fundamentals
2. PRACE and Parallel Computing Basics
3. Unsupervised Clustering and Applications
4. Unsupervised Clustering Challenges & Solutions
5. Supervised Classification and Learning Theory Basics
6. Classification Applications, Challenges, and Solutions
7. Support Vector Machines and Kernel Methods
8. Practicals with SVMs
9. Validation and Regularization Techniques
10. Practicals with Validation and Regularization
11. Parallelization Benefits
12. Cross-Validation Practicals

Day One – beginner

Day Two – moderate

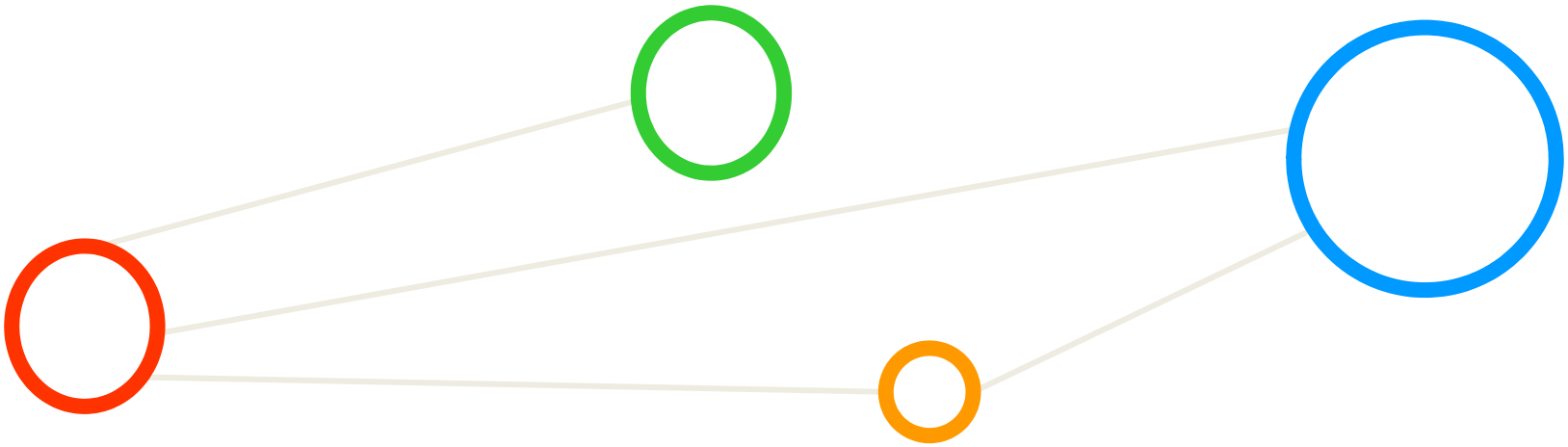
Day Three – expert

# Outline

- Support Vector Machines
  - Term Support Vector Machines Refined
  - Margin as Geometric Interpretation
  - Optimization Problem & Implementation
  - From Hard-margin to Soft-margin
  - Role of Regularization Parameter C

- Kernel Methods
  - Need for Non-Linear Decision Boundaries
  - Non-Linear Transformations
  - Full Support Vector Machines
  - Kernel Trick
  - Polynomial and Radial Basis Function Kernel

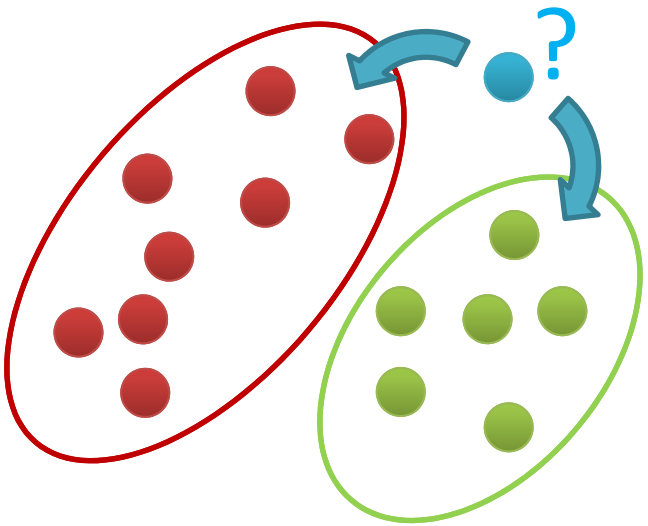
# Support Vector Machines



# Methods Overview – Focus in this Lecture

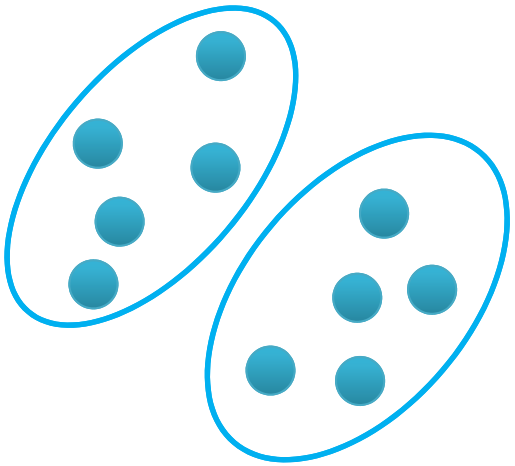
- **Statistical data mining methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction**

## Classification



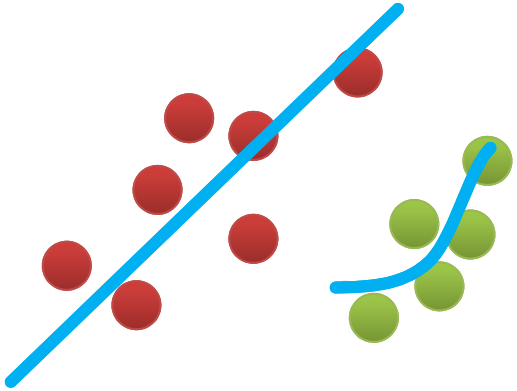

- Groups of data exist
- New data classified to existing groups

## Clustering



- No groups of data exist
- Create groups from data close to each other

## Regression



- Identify a line with a certain slope describing the data

# Term Support Vector Machines Refined

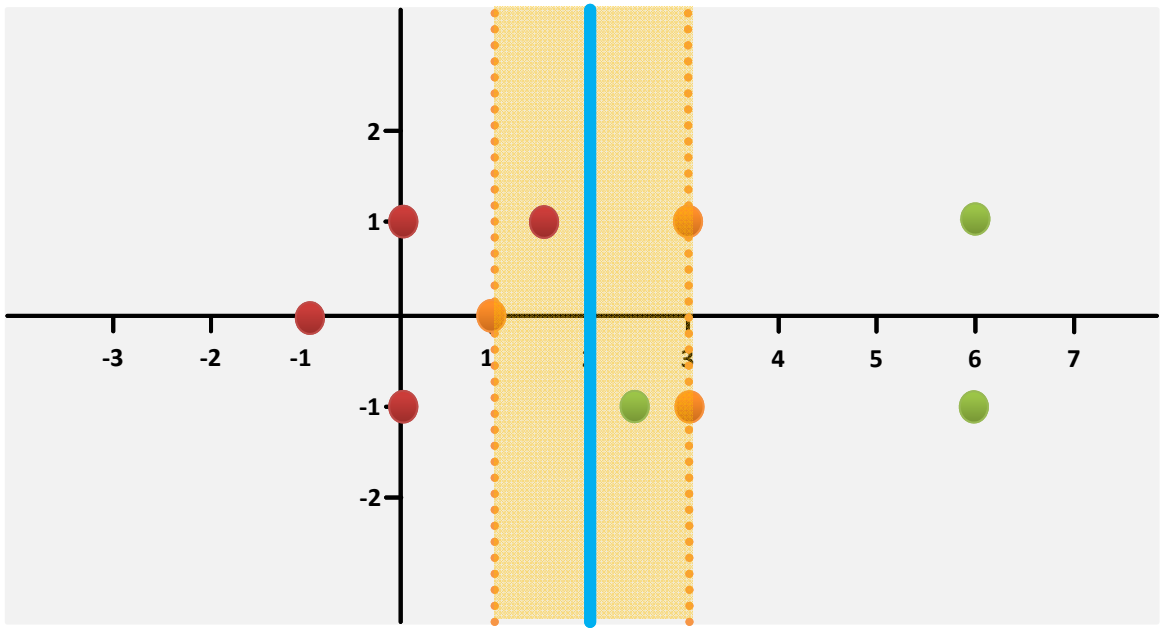
- **Support Vector Machines (SVMs) are a classification technique developed ~1990**
- **SVMs perform well in many settings & are considered as one of the best <u>'out of the box classifiers'</u>**

***[1] An Introduction to Statistical Learning***

- Term detailed refinement into 'three separate techniques'
  - Practice: applications mostly use the SVMs with kernel methods
- 'Maximal margin classifier'
  - A simple and intuitive classifier with a 'best' linear class boundary
  - Requires that data is 'linearly separable'
- 'Support Vector Classifier'
  - Extension to the maximal margin classifier for non-linearly seperable data
  - Applied to a broader range of cases, idea of 'allowing some error'
- 'Support Vector Machines' → Using Non-Linear Kernel Methods
  - Extension of the support vector classifier
  - Enables non-linear class boundaries & via kernels;

# Expected Out-of-Sample Performance for 'Best Line'

- The line with a 'bigger margin' seems to be better – but why?
  - Intuition: chance is higher that a new point will still be correctly classified
  - Fewer hypothesis possible: constrained by sized margin (cf. Lecture 3)
  - Idea: achieving good 'out-of-sample' performance is goal (cf. Lecture 3)




(e.g. better performance compared to PLA technique)

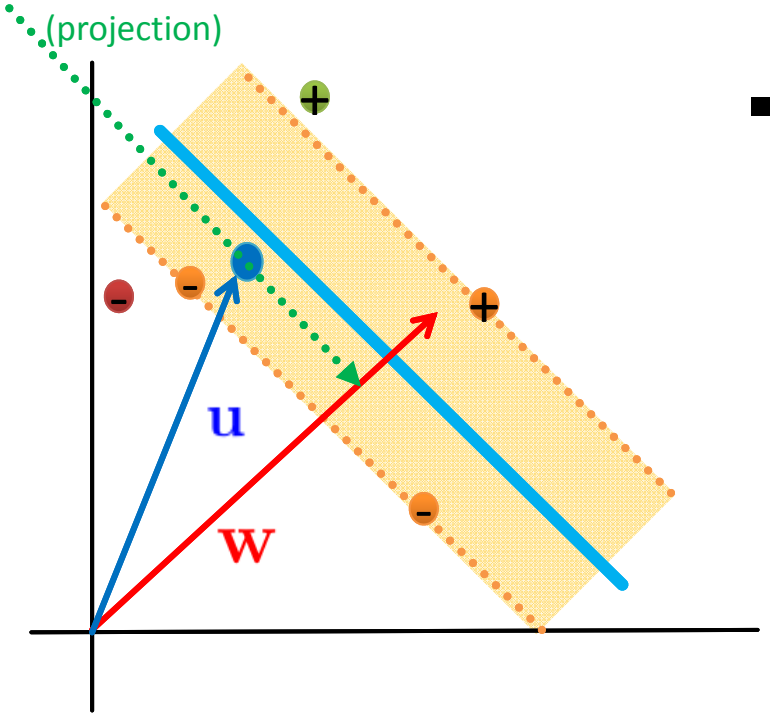
(simple line in a linear setup as intuitive decision boundary)

(Question remains: how we can achieve a bigger margin)

- **Support Vector Machines (SVMs) use maximum margins that will be mathematically established**

# Geometric SVM Interpretation and Setup (1)

- Think ‘simplified coordinate system‘ and use ‘Linear Algebra‘
  - Many other samples are removed (red and green not SVs) - +
  - Vector $\mathbf{w}$ of ‘any length‘ perpendicular to the decision boundary
  - Vector $\mathbf{u}$ points to an unknown quantity (e.g. new sample to classify)
  - Is $\mathbf{u}$ on the left or right side of the decision boundary?




- Dot product $\mathbf{w} \cdot \mathbf{u} \geq C; C = -b$
  - With $\mathbf{u}$ takes the projection on the $\mathbf{w}$
  - Depending on where projection is it is left or right from the decision boundary
  - Simple transformation brings decison rule:

1 $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ → means +

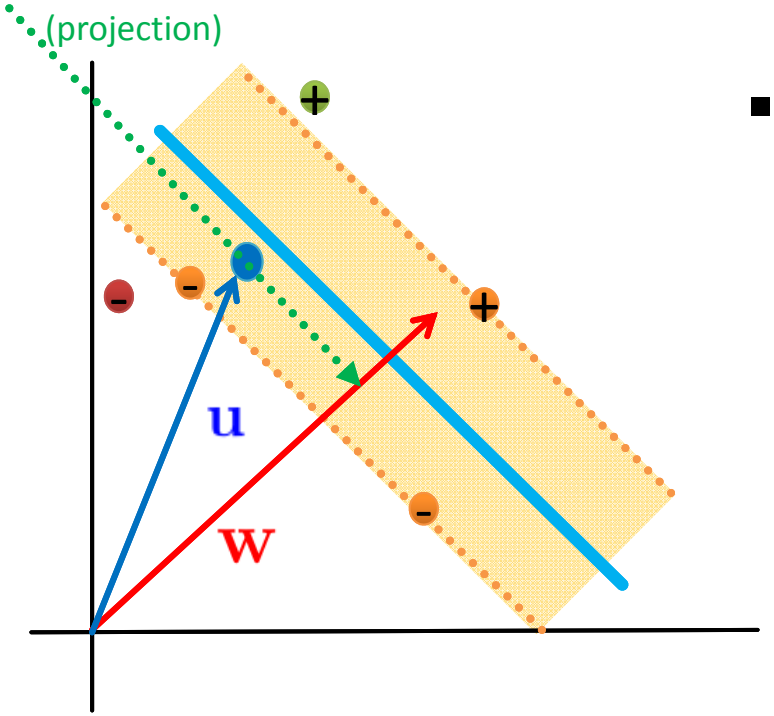
  - (given that b and $\mathbf{w}$ are unknown to us)

(constraints are not enough to fix particular b or w, need more constraints to calculate b or w)

# Geometric SVM Interpretation and Setup (2)

- Creating our constraints to get b or $\mathbf{w}$ computed
  - First constraint set for positive samples ⊕ $\mathbf{w} \cdot \mathbf{x}_+ + b \geq 1$
  - Second constraint set for negative samples ⊖ $\mathbf{w} \cdot \mathbf{x}_- + b \leq 1$
  - For mathematical convenience introduce variables (i.e. labelled samples)

    $y_i = +$ for ⊕ and $y_i = -$ for ⊖




- Multiply equations by $y_i$
  - Positive samples: $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1$
  - Negative samples: $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1$
  - Both same due to $y_i = +$ and $y_i = -$

    (brings us mathematical convenience often quoted)

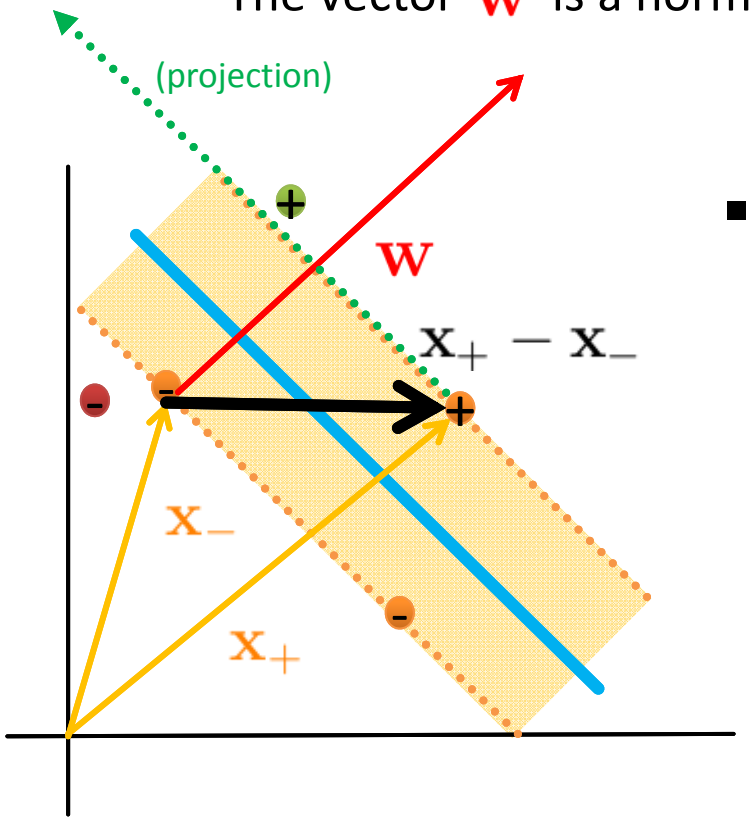
    $$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0$$

    (additional constraints just for support vectors itself helps)

    2 $$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$$

# Geometric SVM Interpretation and Setup (3)

- Determine the 'width of the margin'
  - Difference between positive and negative SVs: $\mathbf{x}_+ - \mathbf{x}_-$
  - Projection of $\mathbf{x}_+ - \mathbf{x}_-$ onto the vector $\mathbf{w}$
  - The vector $\mathbf{w}$ is a normal vector, magnitude is $\|\mathbf{w}\|$




(Dot product of two vectors is a scalar, here the width of the margin)

- Unit vector is helpful for 'margin width'
  - Projection (dot product) for margin width:

$$\mathbf{x}_+ - \mathbf{x}_- \cdot \boxed{\frac{\mathbf{w}}{\|\mathbf{w}\|}} \quad \text{(unit vector)}$$

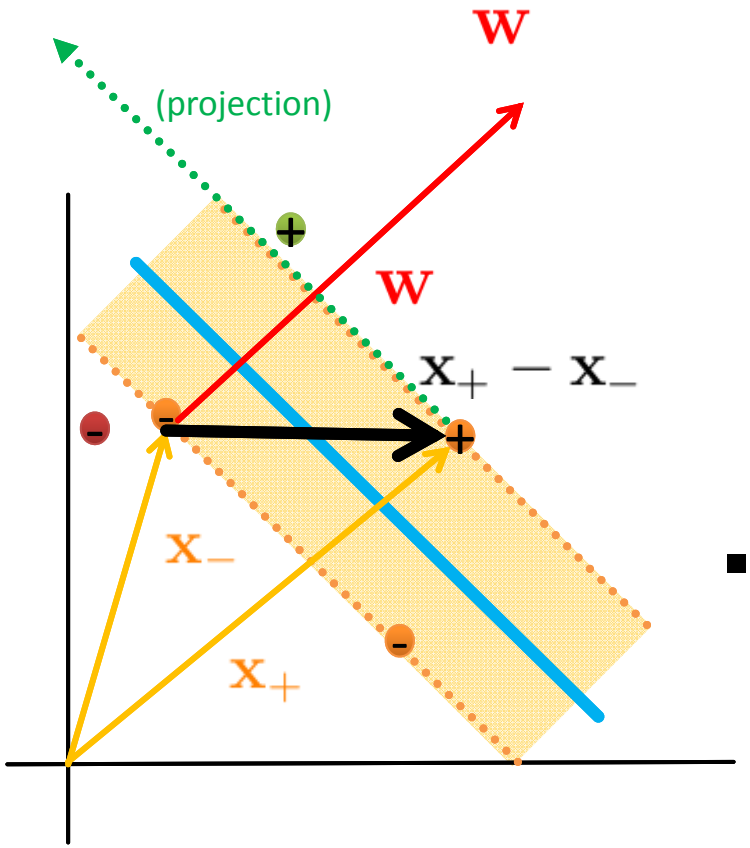
$$1 - b \qquad 1 + b \qquad \frac{2}{\|\mathbf{w}\|} \quad (3)$$

  - When enforce constraint: $y_i = +$ ⊕ $\quad y_i = -$ ⊖

(2) $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$

# Constrained Optimization Steps SVM (1)

- Use ‘constraint optimization‘ of mathematical toolkit
  - Idea is to ‘maximize the width‘ of the margin: $max \frac{2}{\|\mathbf{w}\|}$ (drop the constant 2 is possible here)




$\Rightarrow$ $max \frac{1}{\|\mathbf{w}\|}$ (equivalent)

$\Rightarrow$ $min \|\mathbf{w}\|$ (equivalent for max)

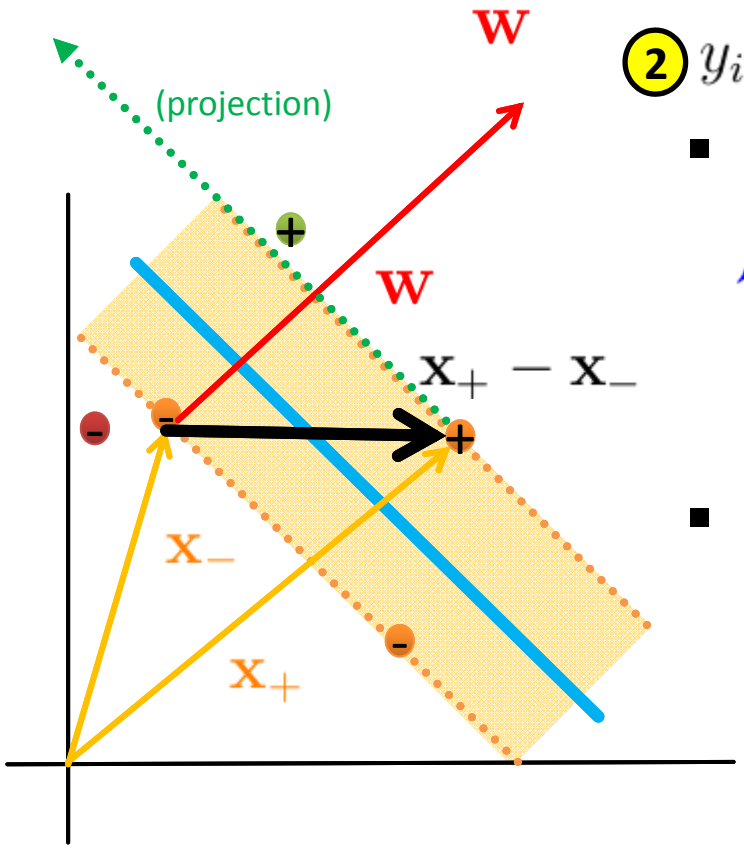
$\Rightarrow$ $min \frac{1}{2} \|\mathbf{w}\|^2$ (mathematical convenience) 3

- Next: Find the extreme values
  - Subject to constraints

2 $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$

# Constrained Optimization Steps SVM (2)

- Use ‘Lagrange Multipliers‘ of mathematical toolkit
  - Established tool in ‘constrained optimization‘ to find function extremum
  - ‘Get rid‘ of constraints by using Lagrange Multipliers 4




2 $y_i(\mathbf{x}_i \cdot \mathbf{w} + b - 1) = 0$

- Introduce a multiplier for each constraint

$$\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$$

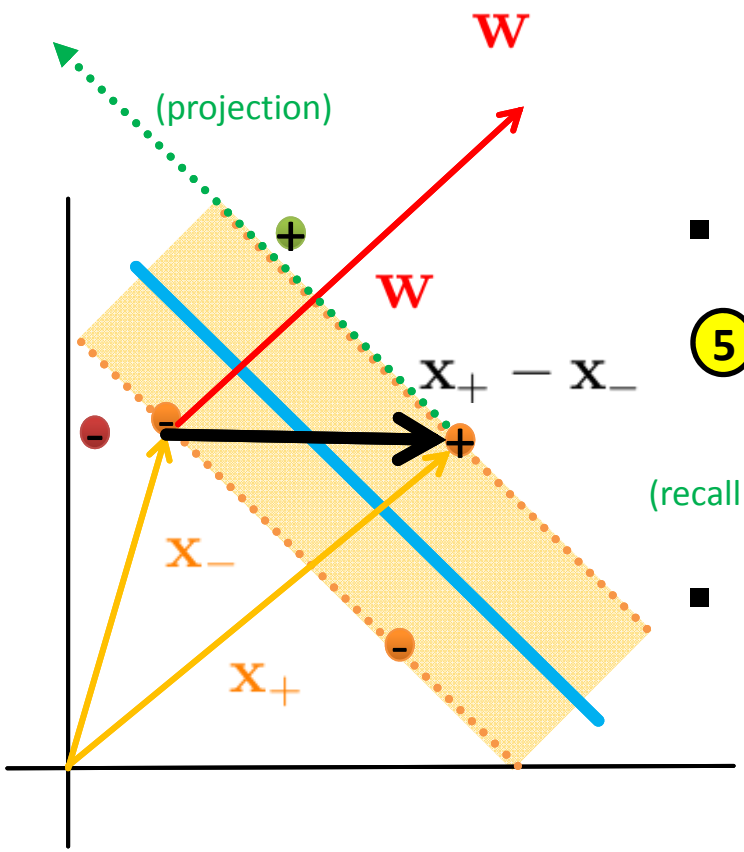
(interesting: non zero for support vectors, rest zero)

- Find derivatives for extremum & set 0
  - But two unknowns that might vary
  - First differentiate w.r.t. $\mathbf{w}$
  - Second differentiate w.r.t. $b$

(derivative gives the gradient, setting 0 means extremum like min)

# Constrained Optimization Steps SVM (3)

- Lagrange gives: $\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$




- First differentiate w.r.t $\mathbf{w}$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \boxed{\sum \alpha_i y_i \mathbf{x}_i} = 0$$

(derivative gives the gradient, setting 0 means extremum like min)

- Simple transformation brings:

5 $$\mathbf{w} = \boxed{\sum \alpha_i y_i \mathbf{x}_i}$$

(i.e. vector is linear sum of samples)

(recall: non zero for support vectors, rest zero → even less samples)

- Second differentiate w.r.t. $b$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y_i = 0 \Rightarrow \sum \alpha_i y_i = 0$$ 5

# Constrained Optimization Steps SVM (4)

- Lagrange gives: $\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$

(plug into)

- Find minimum
  - Quadratic optimization problem
  - Take advantage of 5 $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

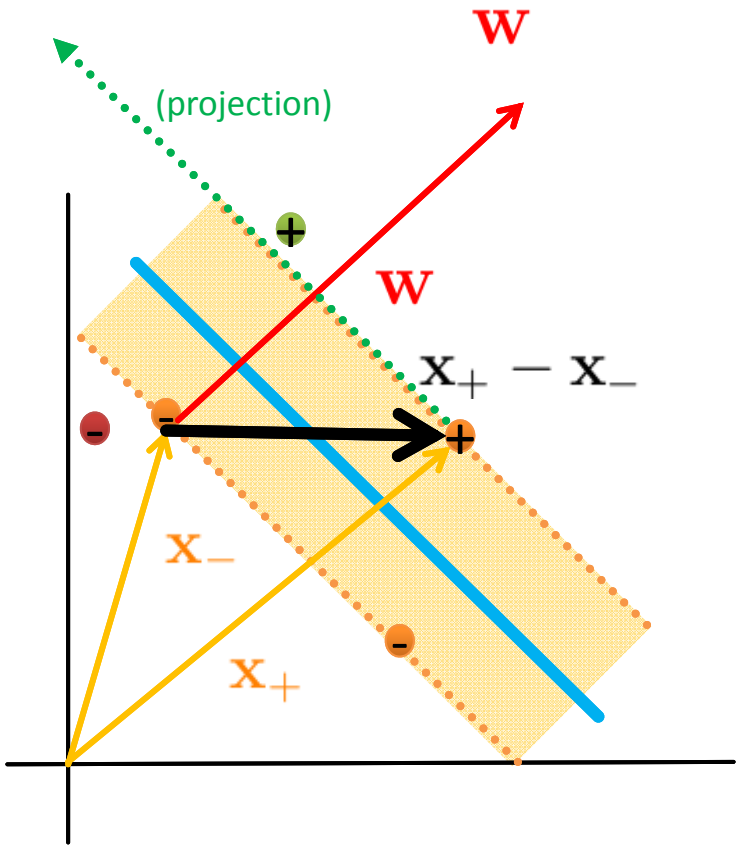
$$\mathcal{L} = \frac{1}{2}(\sum \alpha_i y_i \mathbf{x}_i) \cdot (\sum \alpha_j y_j \mathbf{x}_j)$$

$$- \sum \alpha_i y_i \mathbf{x}_i \cdot (\sum \alpha_j y_j \mathbf{x}_j)$$

$$- \sum \alpha_i y_i b + \sum \alpha_i$$

(b constant in front sum)

5 $\sum \alpha_i y_i = 0$



# Constrained Optimization Steps SVM (5)

- Rewrite formula: $\mathcal{L} = \frac{1}{2}\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$

$$- \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$$

(the same)

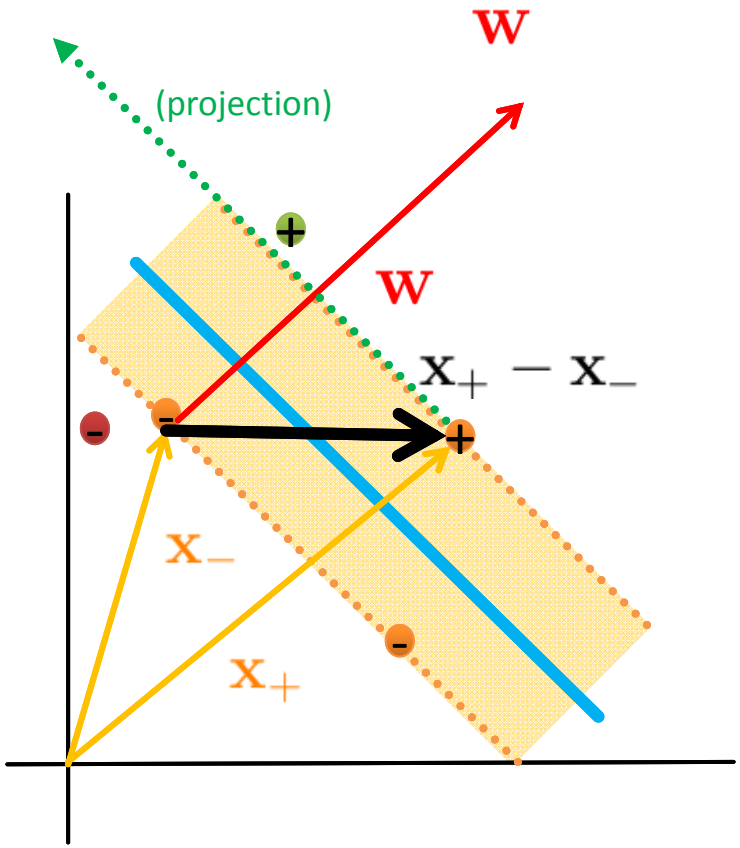
$$\cancel{- \sum \alpha_i y_i b} + \sum \alpha_i$$

(was 0)

(results in)

(optimization depends only on dot product of samples)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (6)$$




- Equation to be solved by some quadratic programming package

# Use of SVM Classifier to Perform Classification

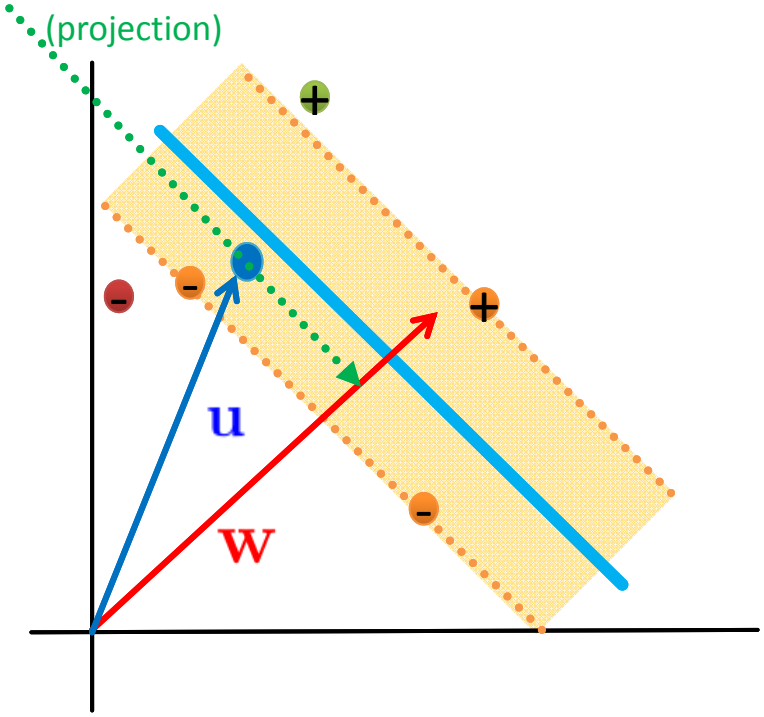
- Use findings for decision rule

(5) $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

(1) $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ +

(decision rule also depends on dotproduct)

$\sum \alpha_i y_i \boxed{\mathbf{x}_i \cdot \mathbf{u}_i} + b \geq 0$ +



# Maximal Margin Classifier – Training Set and Test Set

- Classification technique
  - Given ‘labelled dataset‘
  - Data matrix X (n x p)
  - Training set:
    n training samples
  - p-dimensional space
  - Linearly seperable data
  - Binary classification problem
    (two class classification)
  - Test set:
    a vector x* with test observations

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

(n x p-dimensional vectors)

$$y_1, \ldots, y_n \in \{-1, 1\}$$

(class labels) (two classes)

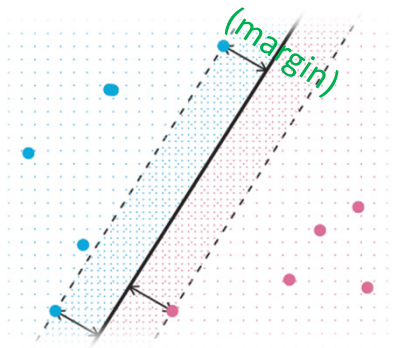
$$x^* = \begin{pmatrix} x_1^* & \ldots & x_p^* \end{pmatrix}^T$$

- **Maximal Margin Classifiers create a seperating hyperplane that seperates the training set samples perfectly according to their class labels following a reasonable way of which hyperplane to use**

***[1] An Introduction to Statistical Learning***

# Maximal Margin Classifier – Reasoning and Margin Term

- Reasoning to pick the ‘best line‘
  - There exists a ‘maximal margin hyperplane‘ (optimal seperating hyperplane)
  - Hyperplane that is ‘farthest away‘ from the training set samples
- Identify the ‘margin‘ itself
  - Compute the ‘perpendicular distance’ (point ‘right angle 90 degrees‘ distance to the plane)
  - From each training sample to a given separating hyperplane
  - The smallest such distance is the ’minimal distance’ from the observations to the hyperplane – the margin
- Identify ‘maximal margin‘
  - Identify the hyperplane that has the ‘farthest minimum distance’ to the training observations
  - Also named the ‘optimal seperating hyperplane‘




- **The maximal margin hyperplane is the seperating hyperplane for which the margin is largest**

***[1] An Introduction to Statistical Learning***

# Maximal Margin Classifier – Margin Performance

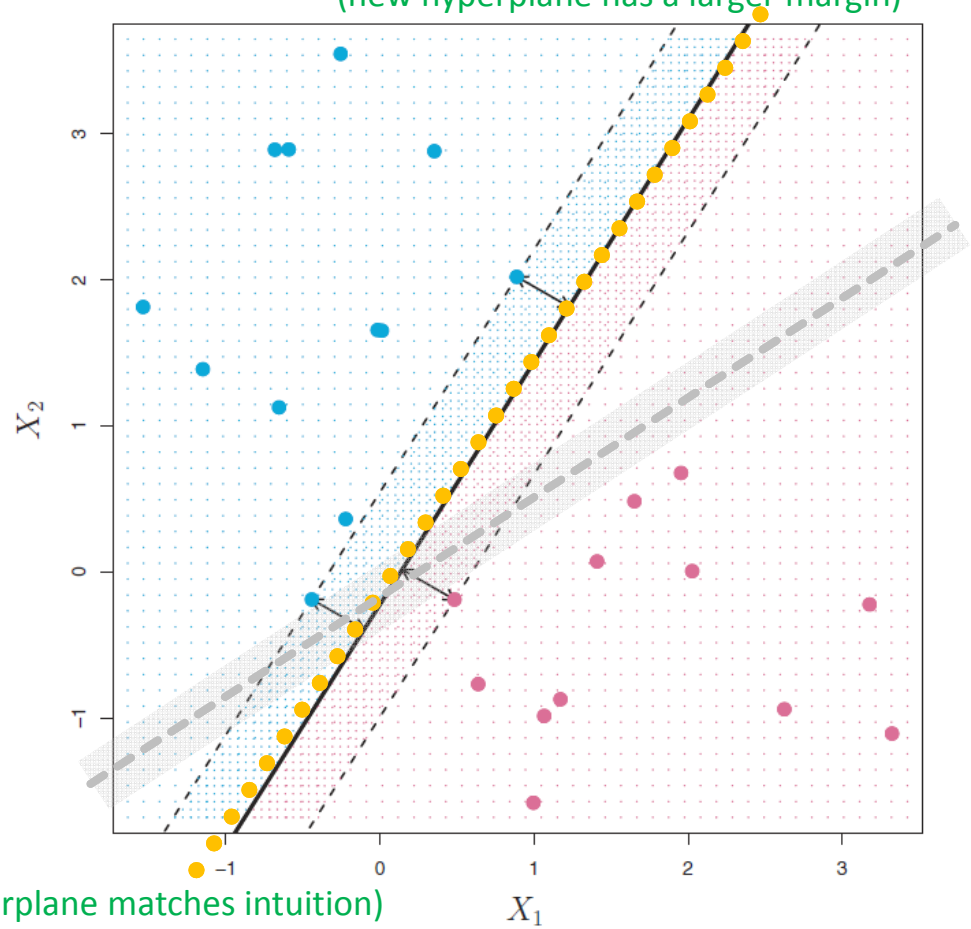
- Classification technique
  - Classify testset samples based on which side of the maximal margin hyperplane they are lying

$$\text{sign of } f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$$

(ßs are the coefficients of the maximal margin hyperplane)

  - Assuming that a classifier that has a large margin on the training data will also have a large margin on the test data (cf. also 'the intuitive notion')
  - Testset samples will be thus correctly classified

(new hyperplane has a larger margin)



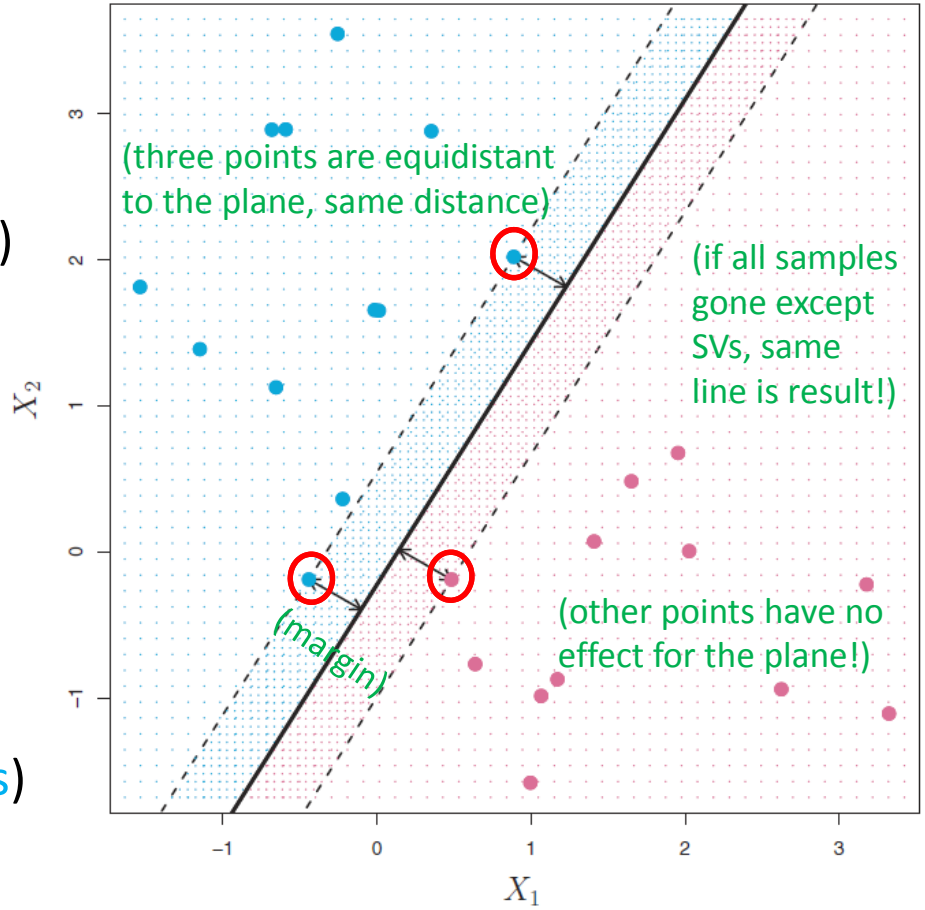

(hyperplane matches intuition)

(Compared to grey hyperplane: a 'greater minimal distance' between the data points and the seperating hyperplane)

***modified from [1] An Introduction to Statistical Learning***

# Maximal Margin Classifier – Support Vector Term

- Observation
  - Three data points lie on the edge of margin (somewhat special data points)
  - Dashed lines indicating the width of the margin (very interesting to know)
  - Margin width is the distance from the special data points to the hyperplane (hyperplane depends directly on small data subset: SV points)



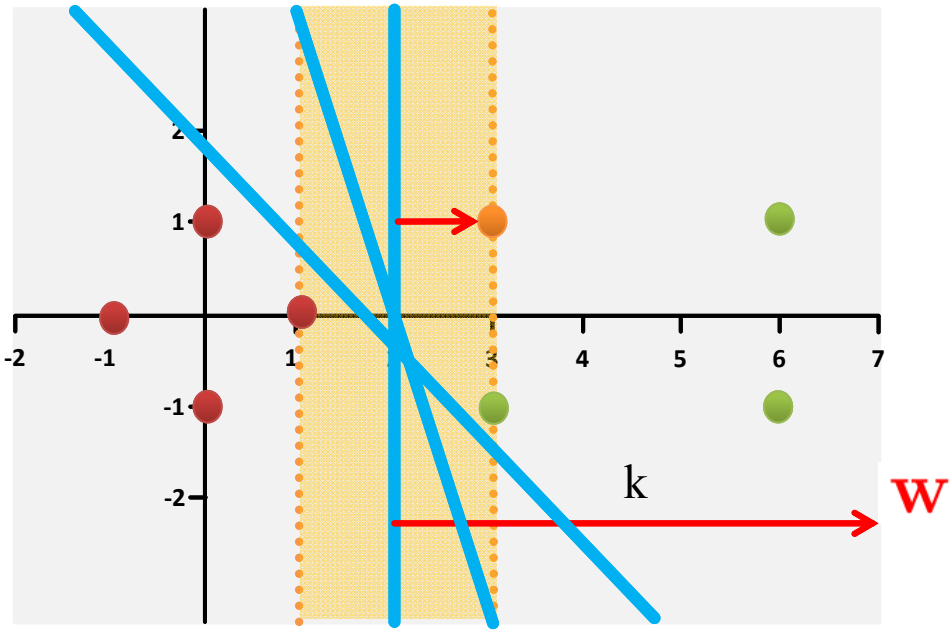

***modified from [1] An Introduction to Statistical Learning***

- **Points that lie on the edge of the margin are named support vectors (SVs) in p-dimensional space**
- **SVs 'support' the maximal margin hyperplane: if SVs are moved → the hyperplane moves as well**

# Maximal Margin Classifier – Optimization and W Vector

- Which weight **w** maximizes the margin?
  - Margin is just a distance from 'a line to a point', goal is to minimize **w**
  - Pick $x_n$ as the nearest data point to the line (or hyper-plane)…

(distance between two dashed planes is 2 / ||w|| )




- Reduce the problem to a 'constraint optimization problem' (vector **w** are the ß coefficients)

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\ M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

(for points on plane w must be 0, interpret k as length of w)

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} = 0 \qquad k(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) = 0 \text{ for any } k \neq 0$$

- **Support vectors achieve the margin and are positioned exactly on the boundary of the margin**

# Maximal Margin Classifier – Optimization and N Samples

- Approach: Maximizing the margin
  - Equivalent to minimize objective function
    (original and modified have same w and b )

$$\min_{w,b} \left\{ \frac{1}{2} \| \mathbf{w} \|^2 \right\}$$

(substituted for plain || w || for mathematical convenience)

subject to $y_i$ ( $\mathbf{w} \cdot x_i$ – b) >= 1

- 'Lagrangian Dual problem' (chain of math turns optimization problem into solving this)
  - Use of already established Lagrangian method :

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m$$

(big data impact, important dot product)

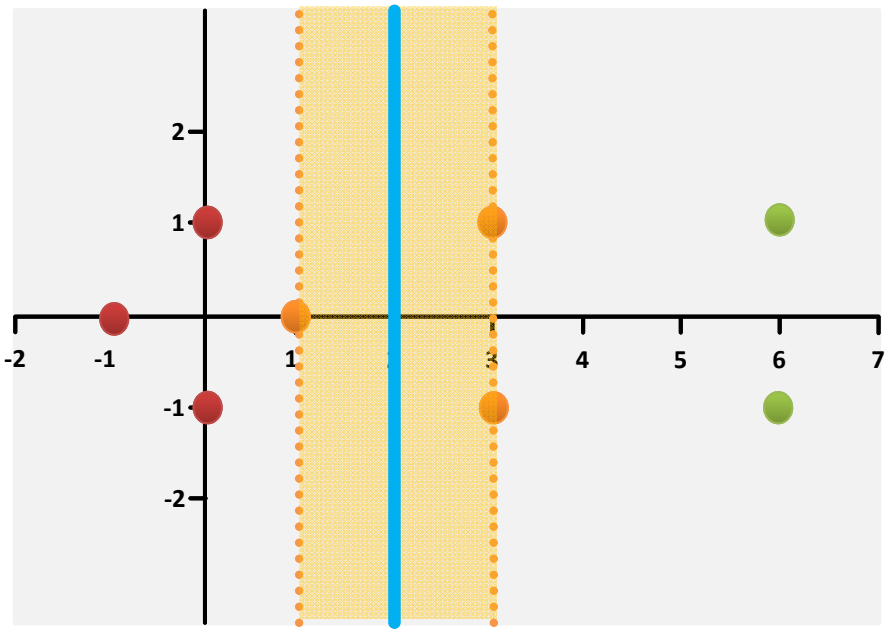
- Interesting properties
  - Simple function: Quadratic in alpha
  - Simple constraints in this optimization problem (not covered here)
  - Established tools exist: Quadratic Programming (qp) (rule of thumb)

- **Practice shows that #N moderate is ok, but large #N ('big data') are problematic for computing**
- **Quadratic programming and computing the solving depends on number of samples N in the dataset**

# Maximal Margin Classifier – Optimization and # SV Impacts

- Interpretation of QP results (vector of alpha is returned)
  - The obtained values of alpha (lagrange multipliers) are mostly 0
  - Only a couple of alphas are > 0 and special: the support vectors (SVs)

(three support vectors create optimal line)




- N x N, usually not sparse
- Computational complexity relies in the following:

(big data challenge) (e.g. all datasets vs. sampling)

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m$$

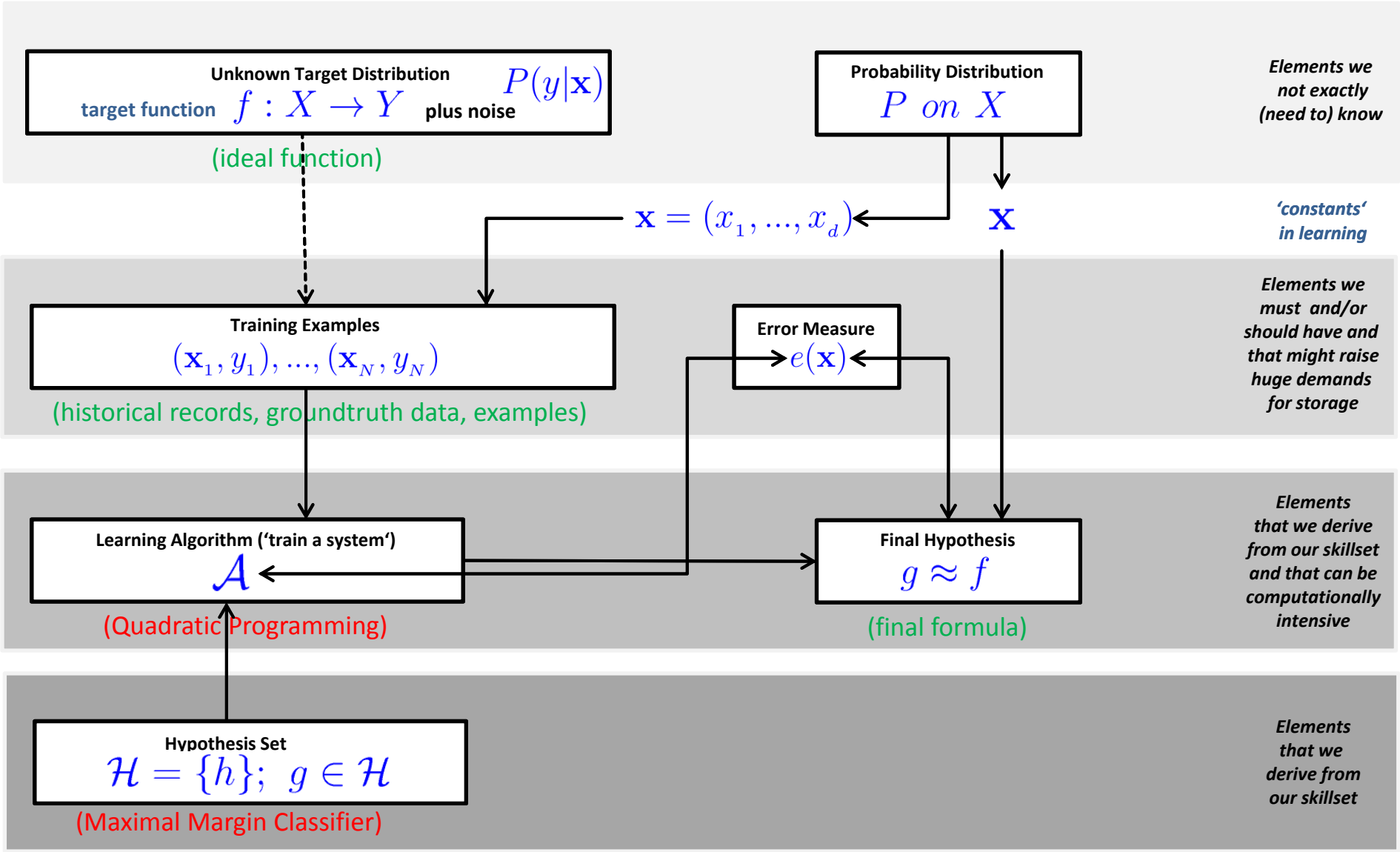
$$\begin{bmatrix} y_1 y_1 x_1^T x_1 & y_1 y_2 x_1^T x_2 & \dots y_1 y_N x_1^T x_N \\ \dots & \dots & \ddots \\ y_N y_1 x_N^T x_1 & y_N y_2 x_N^T x_2 & \dots y_N y_N x_N^T x_N \end{bmatrix}$$

(quadratic coefficients, alphas are result from QP)

(rule of thumb)

- **Generalization measure: #SVs as 'in-sample quantity' → 10SVs/1000 samples ok, 500SVs/1000 bad**
- **Reasonsing towards overfitting due to a large number of SVs (fit many, small margin, gives bad $E_{out}$)**

# Solution Tools: Maximal Margin Classifier & QP Algorithm



# Maximal Margin Classifier – Solving and Limitations

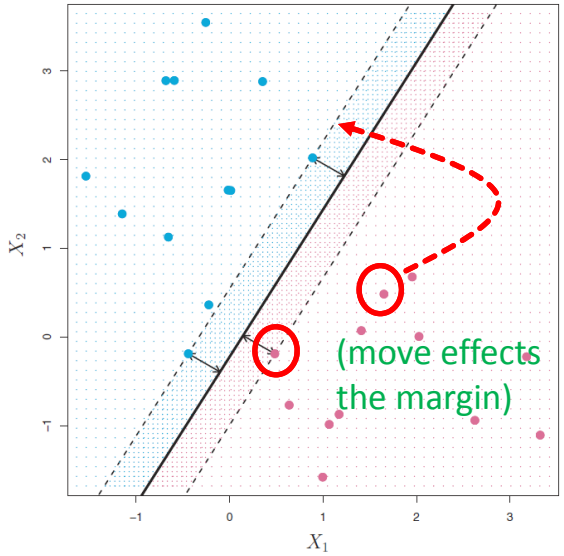
- **Solving constraint optimization problem chooses coefficients that maximize M & gives hyperplane**
- **Solving this problem efficiently is possible techniques like sequential minimal optimization (SMO)**
- **Maximal margin classifiers use a hard-margin & thus only work with exact linearly seperable data**

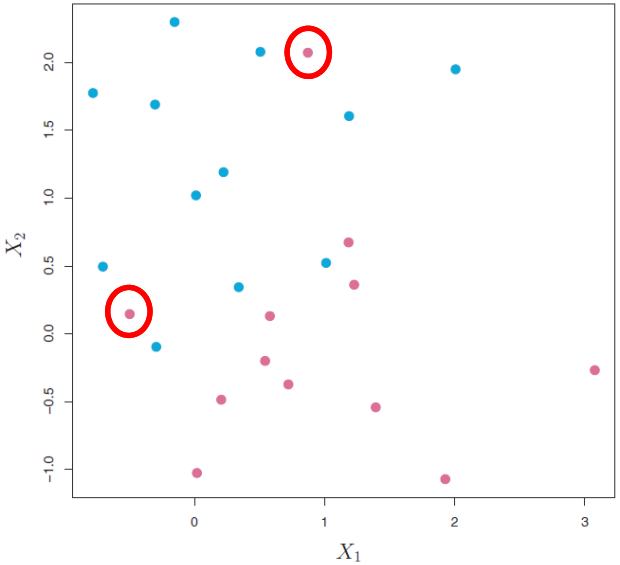
***modified from [1] An Introduction to Statistical Learning***

- Limitation
  - Non linearly separable data (given mostly in practice)
  - Optimization problem has no solution M > 0 (think point moves over plane)
  - No separating hyperplane can be created (classifier can not be used)

(no error allowed, a 'hard margin')

(allow some error the margin will be bigger, … maybe better $E_{out}$)




(no exact seperation possible)

(… but with allowing some error maybe, a 'soft margin'…)

# Exercises – Submit piSVM & Rome (linear)

# JUROPA3 System – SSH Login

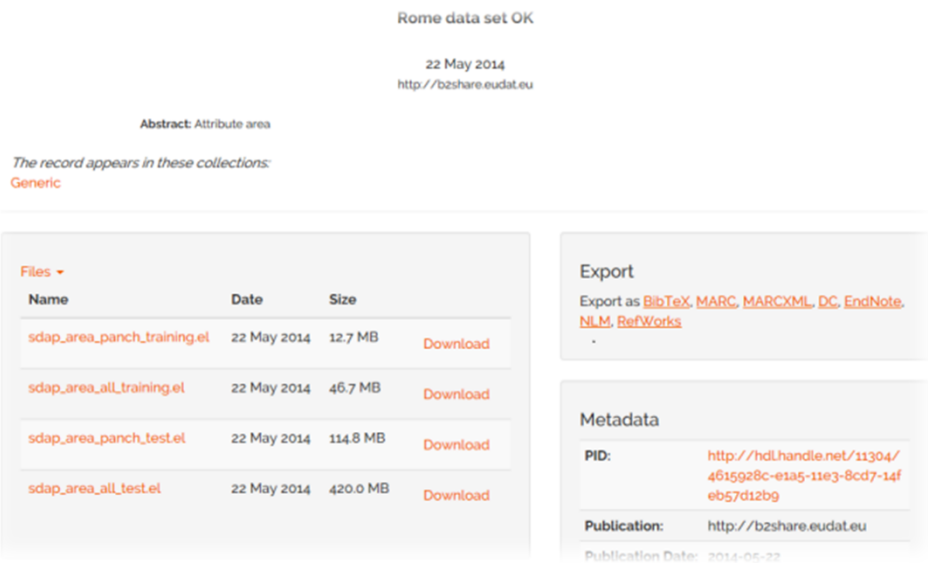
- Use your account train004 - train050
- Windows: use mobaxterm or putty
- UNIX: ssh trainXYZ@juropa3.zam.kfa-juelich.de
- Example

```
adminuser@linux-8djg:~> ssh train001@juropa3.zam.kfa-juelich.de
Last login: Wed Mar 15 09:08:55 2017 from 172.18.57.177
*********************************************************************
*                            Welcome to JUROPA3                     *
*                                                                   *
* Information about the system, latest changes, user documentation and FAQs:  *
*                  http://www.fz-juelich.de/ias/jsc/juropa-3        *
*********************************************************************
[train001@j3l02 ~]$ hostname -f
j3l02.juropa3
```

➢ **Remember to use your own trainXYZ account in order to login to the JUROPA3 system**

# Rome Remote Sensing Dataset (cf. Lecture 6)

- Data is already available in the tutorial directory




(persistent handle link for publication into papers)

B2SHARE
Store and Share Research Data

***[3] Rome Image dataset***

```
[train001@jrl12 rome]$ pwd
/homea/hpclab/train001/data/rome
[train001@jrl12 rome]$ ls -al
total 580256
drwxr-xr-x 2 train001 hpclab       512 Jan 14 21:52 .
drwxr-xr-x 6 train001 hpclab       512 Jan 14 21:47 ..
-rw-r--r-- 1 train001 hpclab 419974873 Dec 22  2016 sdap_area_all_test.el
-rw-r--r-- 1 train001 hpclab  46652874 Dec 22  2016 sdap_area_all_training.el
-rw-r--r-- 1 train001 hpclab 114763982 Dec 22  2016 sdap_area_panch_test.el
-rw-r--r-- 1 train001 hpclab  12745692 Dec 22  2016 sdap_area_panch_training.el
```

# HPC Environment – Modules Revisited

- Module environment tool
  - Avoids to manually setup environment information for every application
  - Simplifies shell initialization and lets users easily modify their environment
- Module avail
  - Lists all available modules on the HPC system (e.g. compilers, MPI, etc.)
- Module spider
  - Find modules in the installed set of modules and more information
- Module load → needed before piSVM run
  - Loads particular modules into the current work environment, E.g.:

```
[train001@jrl05 jsc_mpi]$ module load Intel/2018.1.163-GCC-5.4.0
[train001@jrl05 jsc_mpi]$ module load IntelMPI/2018.1.163
```

# Parallel & Scalable PiSVM – Parameters

```
[train001@j3l02 pisvm-1.2.1]$ ./pisvm-train
Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
        0 -- C-SVC
        1 -- nu-SVC
        2 -- one-class SVM
        3 -- epsilon-SVR
        4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/k)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 40)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight: set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-o n: max. size of working set
-q n: max. number of new variables entering working set
flags:
-D: Assume the feature vectors are dense (default: sparse)
```

- **C-SVC: The cost (C) in this case refers to a soft-margin specifying how much error is allowed and thus represents a regularization parameter that prevents overfitting → more details tomorrow**
- **nu-SVC: nu in this case refers to values between 0 and 1 and thus represents a lower and upper bound on the number of examples that are support vectors and that lie on the wrong side of the hyperplane**

***[2] www.big-data.tips, 'SVM Train'***

# Training Rome on JUROPA3 – Job Script (linear)

- Use Rome Dataset with paralle & scalable piSVM tool
  - Parameters are equal to the serial libsvm and some additional parameters for paralellization

```
#!/bin/bash -x
#SBATCH--nodes=2
#SBATCH--ntasks=48
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=01:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=train-rome-lin-2-48-24

### location executable
PISVM=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-train

### location data
TRAINDATA=/homea/hpclab/train001/data/rome/sdap_area_all_training.el

### submit
srun $PISVM -D -o 1024 -q 512 -c 100 -g 8 -t 0 -m 1024 -s 0 $TRAINDATA
```

- **Note the tutorial reservation with –reservation=ml-hpc-2 just valid for today on JUROPA3**

# Testing Rome on JUROPA3 – Job Script (linear)

- Use Rome Dataset with paralle & scalable piSVM tool
  - Parameters are equal to the serial libsvm and some additional parameters for paralellization

```
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=01:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=pred-rome-2-48-24

### location executable
PISVMPRED=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-predict

### location data
TESTDATA=/homea/hpclab/train001/data/rome/sdap_area_all_test.el

### trained model data
MODELDATA=/homea/hpclab/train001/tools/pisvm-1.2.1/sdap_area_all_training.el.model

### submit
srun $PISVMPRED $TESTDATA $MODELDATA results.txt
```

- **Note the tutorial reservation with –reservation=ml-hpc-2 just valid for today on JUROPA3**

# Testing Rome on JUROPA3 – Check Outcome

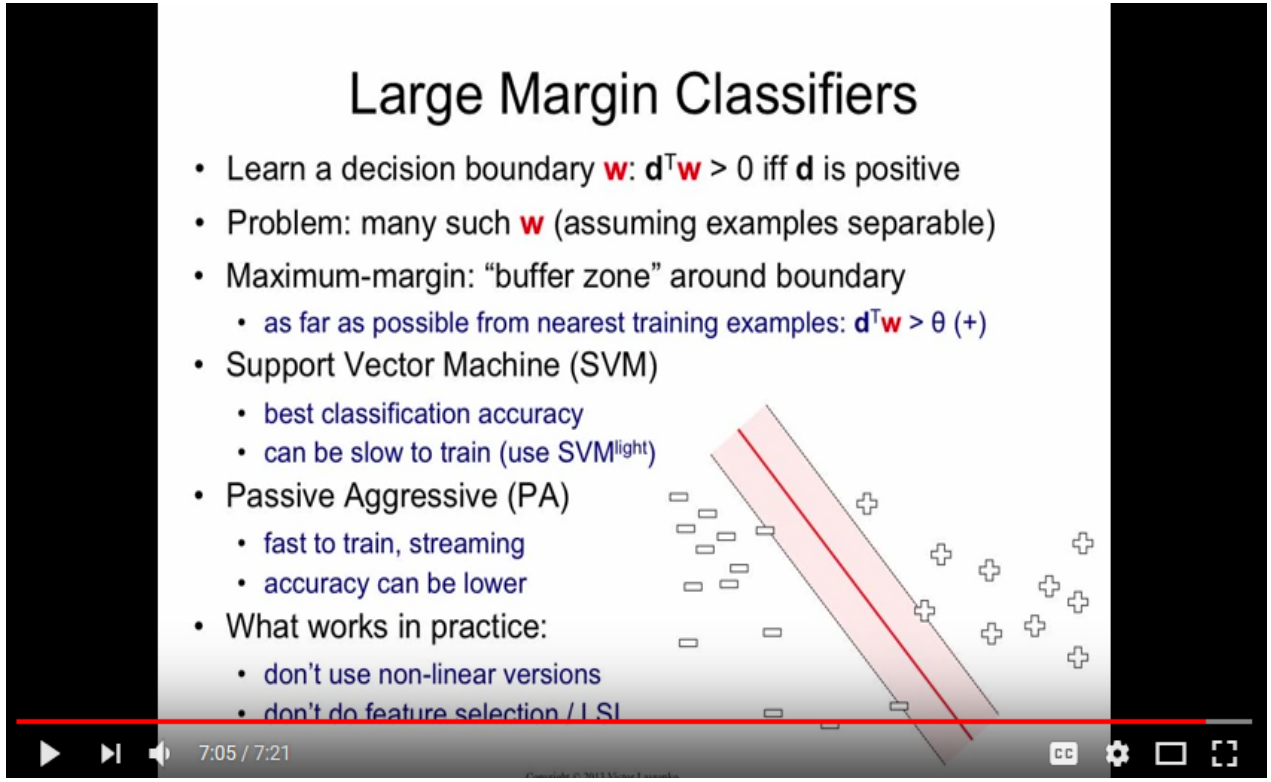
- The output of the training run is a model file
  - Used for input for the testing/predicting phase
  - In-sample property → Support vectors of the model

```
### trained model data
MODELDATA=/homea/hpclab/train001/tools/pisvm-1.2.1/sdap_area_all_training.el.model
```

- The output of the testing/predicting phase is accuracy
  - Accuracy measurement of model performance (cf. Lecture 1)
  - The job output file consists of a couple of lines:

```
[train001@j3l02 pisvm-1.2.1]$ more mpi-out.15244542
Accuracy = 91.5994% (639235/697859) (classification)
Mean squared error = 1.04794 (regression)
Squared correlation coefficient = 0.835385 (regression)
```

# [Video] Maximum Margin




*[4] YouTube Video, Text Classification 2: Maximum Margin Hyperplane'*

# Term Support Vector Machines – Revisited

- **Support Vector Machines (SVMs) are a classification technique developed ~1990**
- **SVMs perform well in many settings & are considered as one of the best <u>'out of the box classifiers'</u>**

***[1] An Introduction to Statistical Learning***

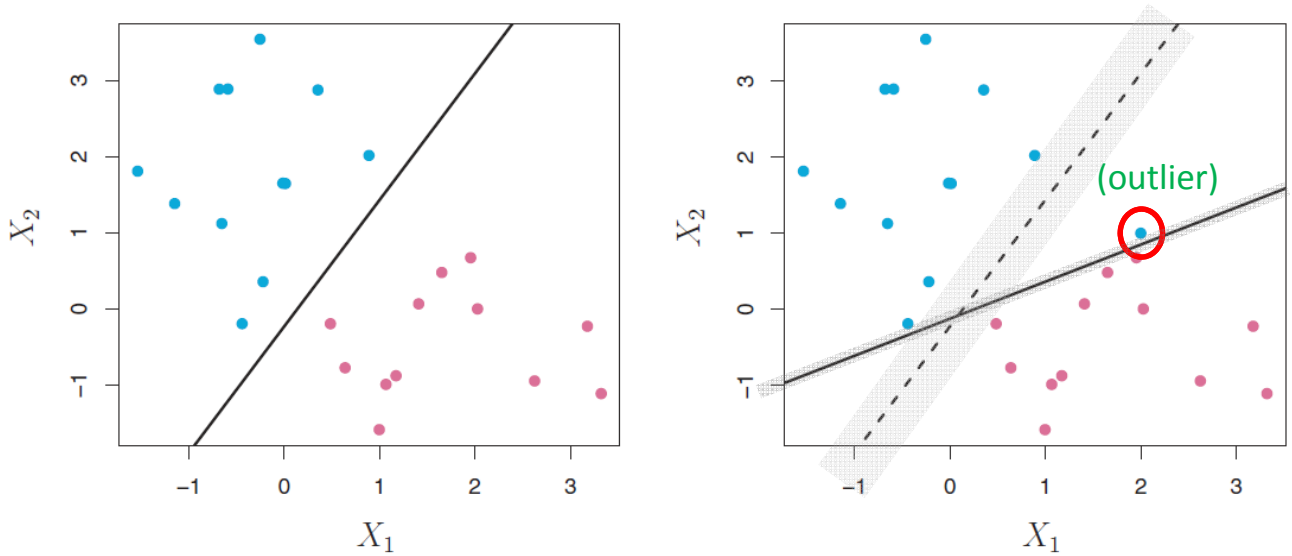
- Term detailed refinement into 'three separate techniques'
  - Practice: applications mostly use the SVMs with kernel methods
- 'Maximal margin classifier'
  - A simple and intuitive classifier with a 'best' linear class boundary
  - Requires that data is 'linearly separable'
- 'Support Vector Classifier'
  - Extension to the maximal margin classifier for non-linearly seperable data
  - Applied to a broader range of cases, idea of 'allowing some error'
- 'Support Vector Machines' → Using Non-Linear Kernel Methods
  - Extension of the support vector classifier
  - Enables non-linear class boundaries & via kernels;

# Support Vector Classifiers – Motivation

- **Support Vector Classifiers develop hyperplanes with soft-margins to achieve better performance**
- **Support Vector Classifiers aim to avoid being sensitive to individual observations (e.g. outliers)**

***[1] An Introduction to Statistical Learning***

- Approach
  - Generalization of the ‘maximal margin classifier‘ (get most & but not all training data correcly classified)
  - Include non-seperable cases with a soft-margin (almost instead of exact)
  - Being more robust w.r.t. extreme values (e.g. outliers) allowing small errors



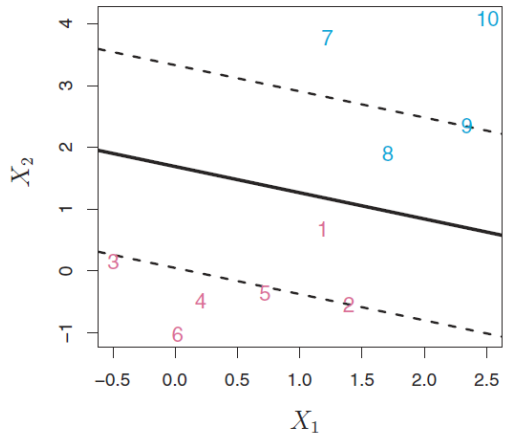

(significant reduction of the maximal margin)

(overfitting, cf Lecture 10: maximal margin classifier & hyperplane is very sensitive to a change of a single data point)

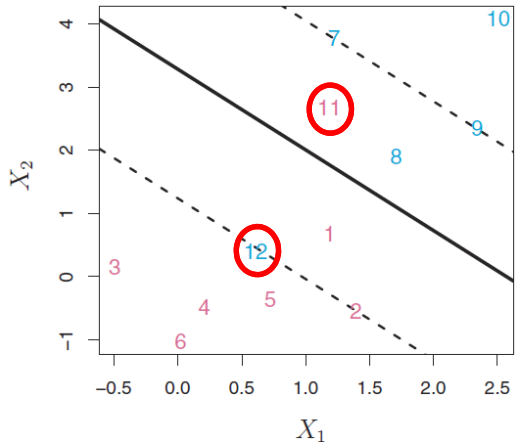
# Support Vector Classifiers – Modified Technique Required

- Previous classification technique reviewed
  - Seeking the largest possible margin, so that every data point is…
  - … on the correct side of the hyperplane
  - … on the correct side of the margin
- Modified classification technique
  - Enable ‘violations of the hard-margin‘ to achieve ‘soft-margin classifier‘
  - Allow violations means: allow some data points to be…
  - … on the incorrect side of the margin
  - … even on the incorrect side of the hyperplane (which leads to a misclassification of that point)

(incorrect side of margin)




(incorrect side of hyperplane)




***[1] An Introduction to Statistical Learning***

(SVs refined: data points that lie directly on the margin or on the wrong side of the margin for their class are the SVs → affect support vector classifer)

# Support Vector Classifier – Optimization Problem Refined

- Optimization Problem
  - Still maximize margin M
  - Refining constraints to include violoation of margins
  - Adding slack variables $\epsilon_1, \ldots, \epsilon_n$

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

(allow datapoints to be on the wrong side of the margin or hyperplane)

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$ (slightly violate the Margin)

$$\boxed{\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C}$$

(C is used here to bound the error)

(C is a nonnegative tuning parameter useful for regularization, cf. Lecture 10, picked by cross-validation method)

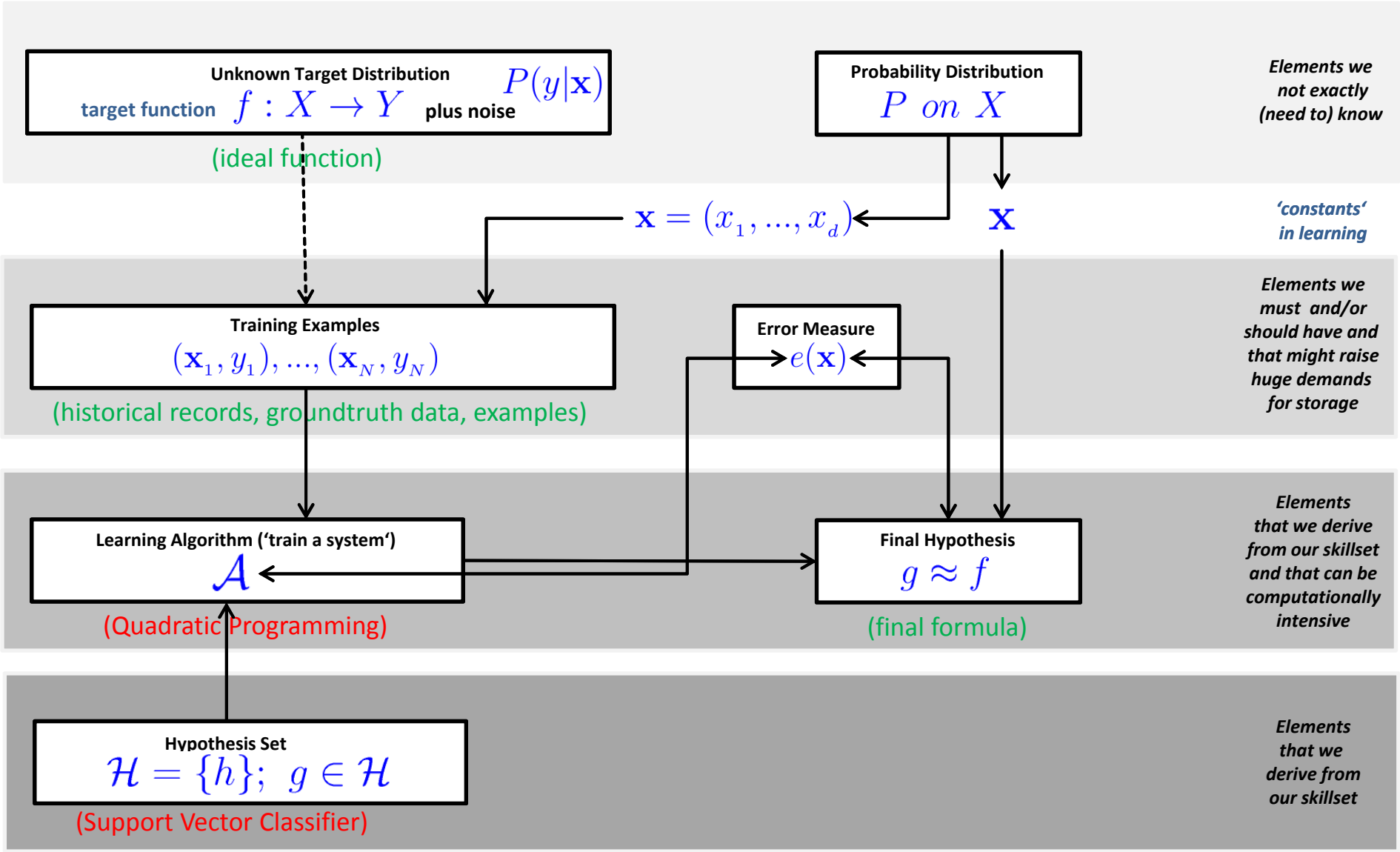
- C Parameter & slacks
  - C bounds the sum of the slack variables $\epsilon_1, \ldots, \epsilon_n$

***[1] An Introduction to Statistical Learning***

- **C determines the number & severity of violations that will be tolerated to margin and hyperplane**
- **Interpret C as budget (or costs) for the amount that the margin can be violated by n data points**

➢ **Lecture 9 provides details on validation methods that provides a value for C in a principled way**

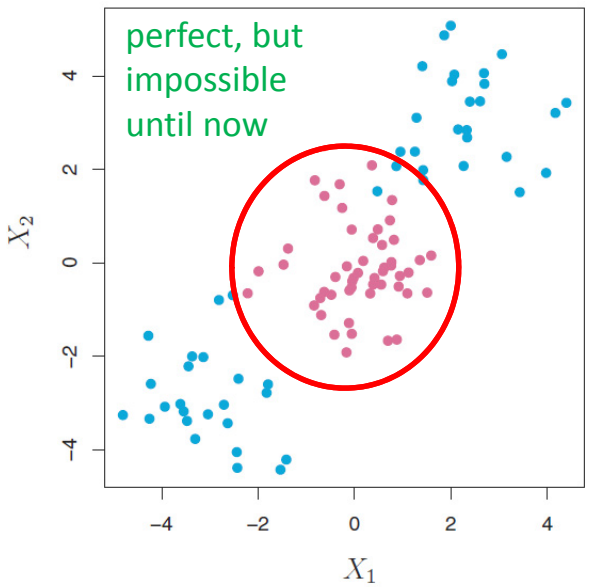
# Solution Tools: Support Vector Classifier & QP Algorithm

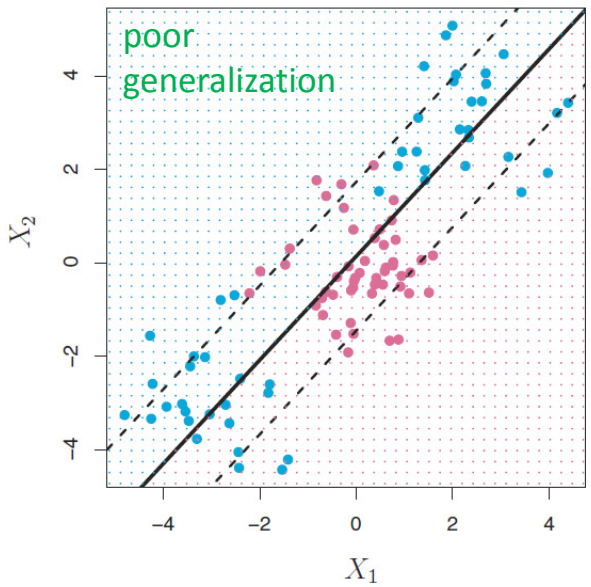


# Support Vector Classifier – Solving & Limitations

- **Solving constraint optimization problem chooses coefficients that maximize M & gives hyperplane**
- **Solving this problem efficiently is possible due to sequential minimal optimization (SMO)**
- **Support vector classifiers use a soft-margin & thus work with slightly(!) non-linearly seperable data**

- Limitation: Still linear decision boundary…
  - Non linearly separable where soft-margins are no solution
  - Support Vector Classifier can not establish a non-linear boundary



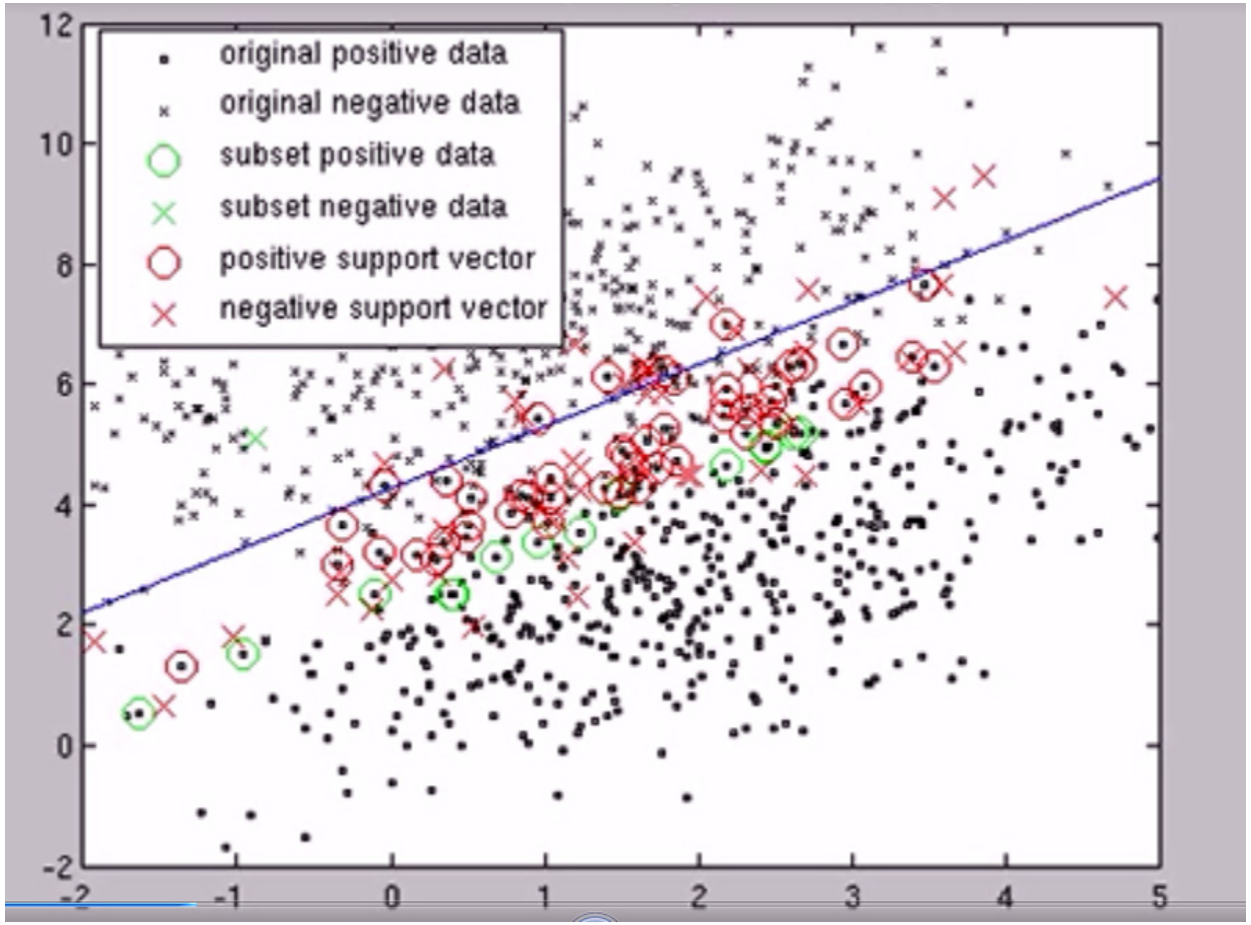




(a linear decision boundary by support vector classifier is not an option)

(… but maybe there are more advanced techniques that help…)

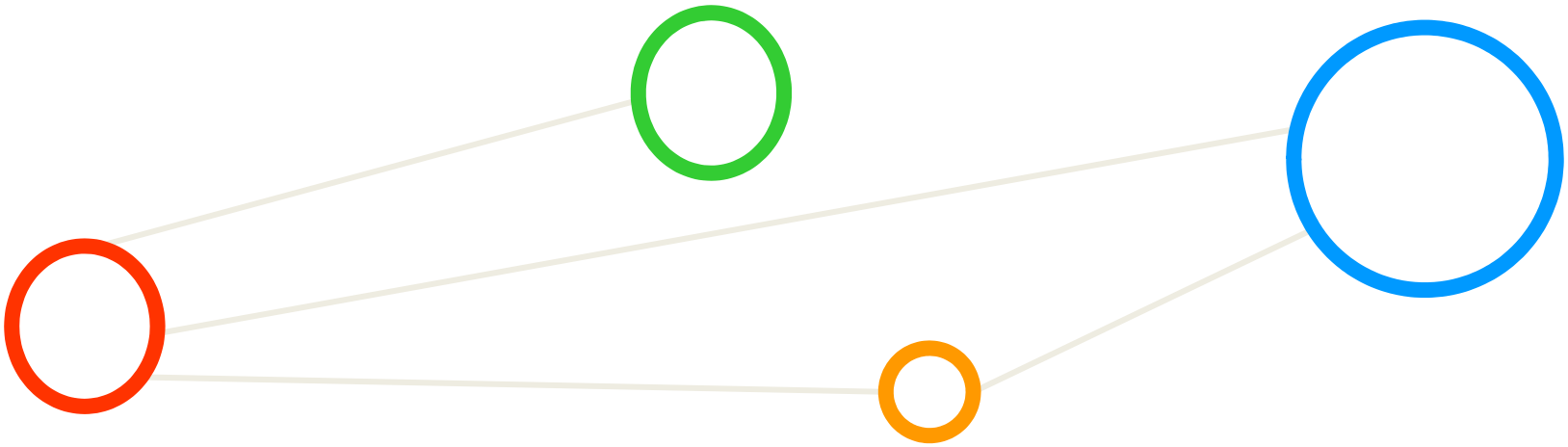
***modified from [1] An Introduction to Statistical Learning***

# [Video] Training Process of Support Vector Machines



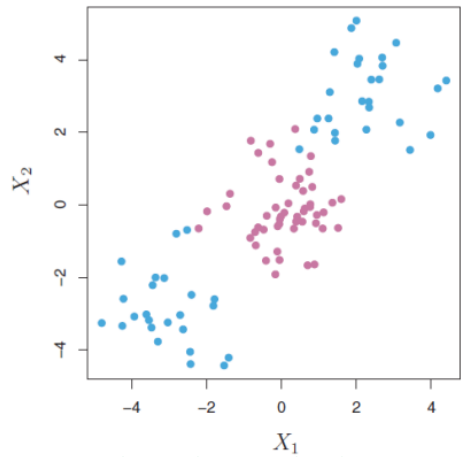

*[5] YouTube Video, 'Cascade SVM'*

# Kernel Methods



# Need for Non-linear Decision Boundaries

- Lessons learned from practice
  - Scientists and engineers are often faced with non-linear class boundaries
- Non-linear transformations approach
  - Enlarge feature space (computationally intensive)
  - Use quadratic, cubic, or higher-order polynomial functions of the predictors
- Example with Support Vector Classifier




(time invest: mapping done by explictly carrying out the map into the feature space)

$X_1, X_2, \ldots, X_p$ (previously used p features)

$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2$ (new 2p features)

(decision boundary is linear in the enlarged feature space)

(decision boundary is non-linear in the original feature space with q(x) = 0 where q is a quadratic polynomial)

$$\underset{\beta_0,\beta_{11},\beta_{12},\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

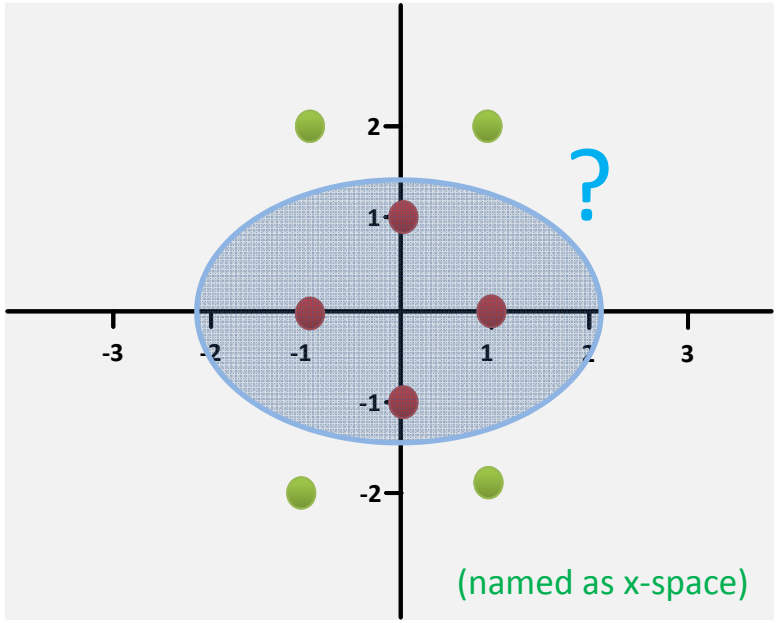
$$\text{subject to } y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_{j1}x_{ij} + \sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right) \geq M(1-\epsilon_i)$$

$$\sum_{i=1}^{n}\epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2 = 1$$

***[1] An Introduction to Statistical Learning***

# Understanding Non-Linear Transformations (1)

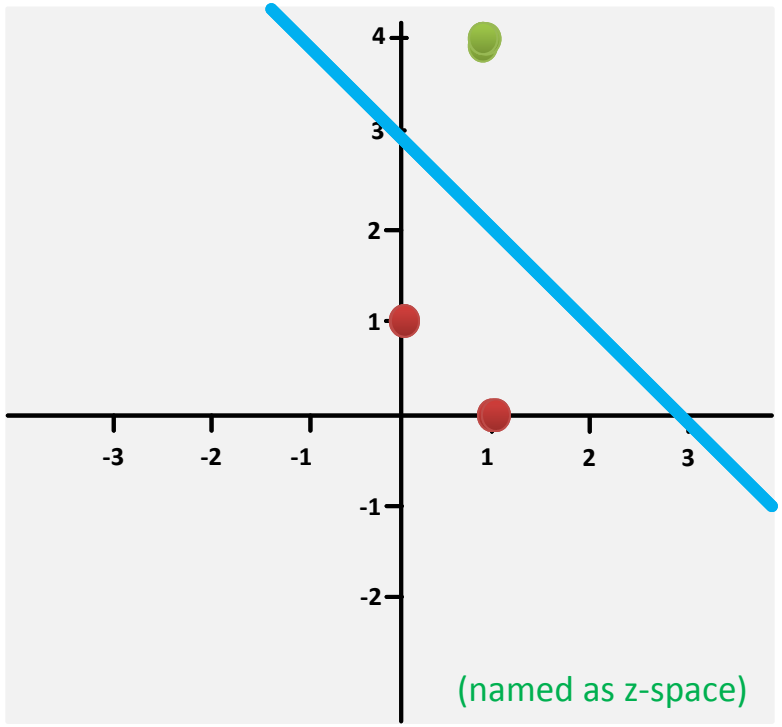
- Example: ‘Use measure of distances from the origin/centre‘
- Classification (still linear models applicable)
  - (1) new point; (2) transform to z-space; (3) classify it with e.g. perceptron




(also called input space)

$\Phi$

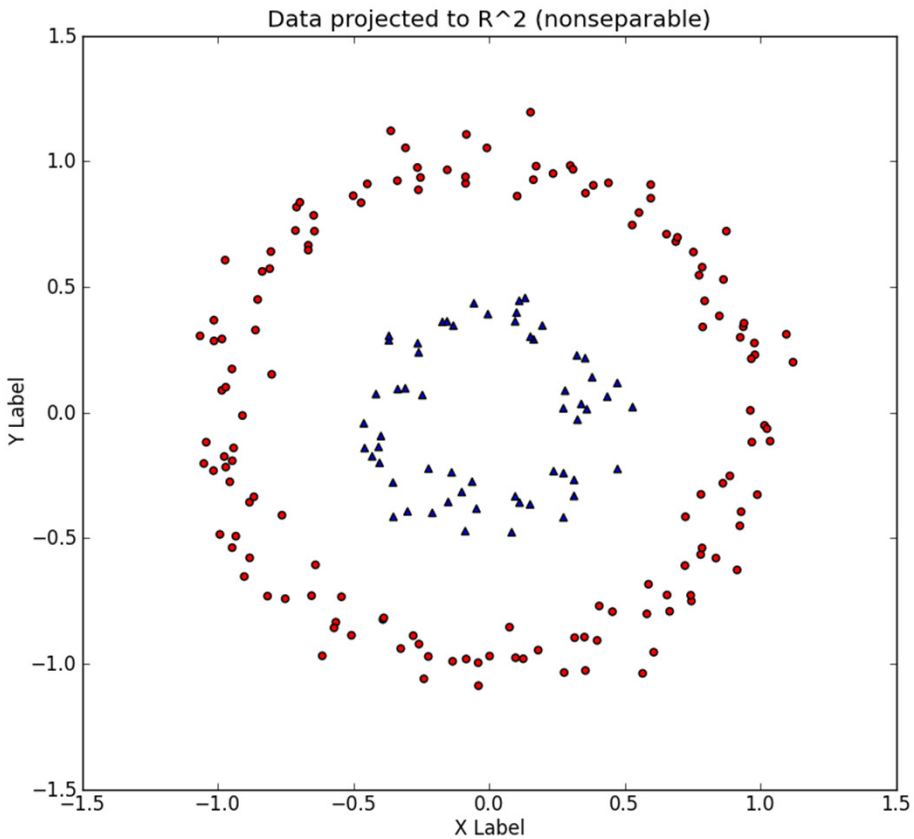
(‘changing constants‘)

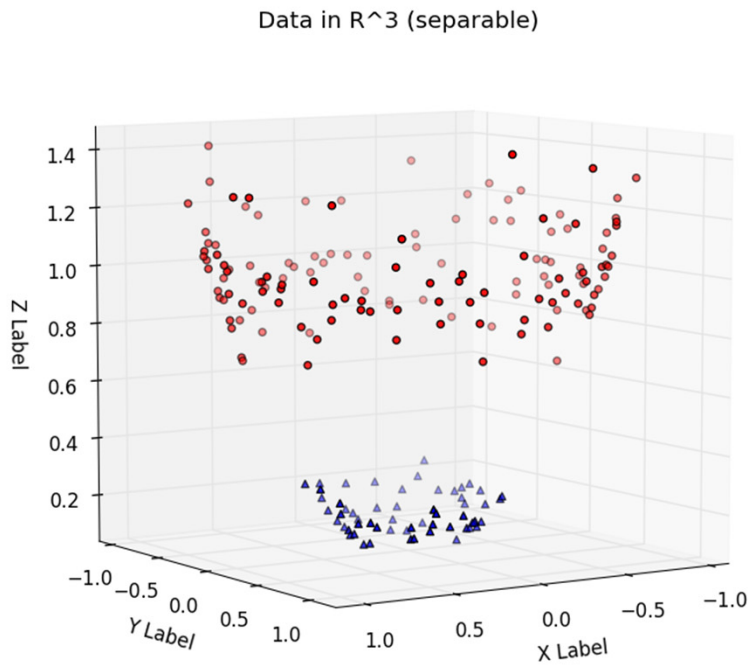



(also called feature space)

# Understanding Non-Linear Transformations (2)

- Example: From 2 dimensional to 3 dimensional: $[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$
  - Much higher dimensional can cause memory and computing problems



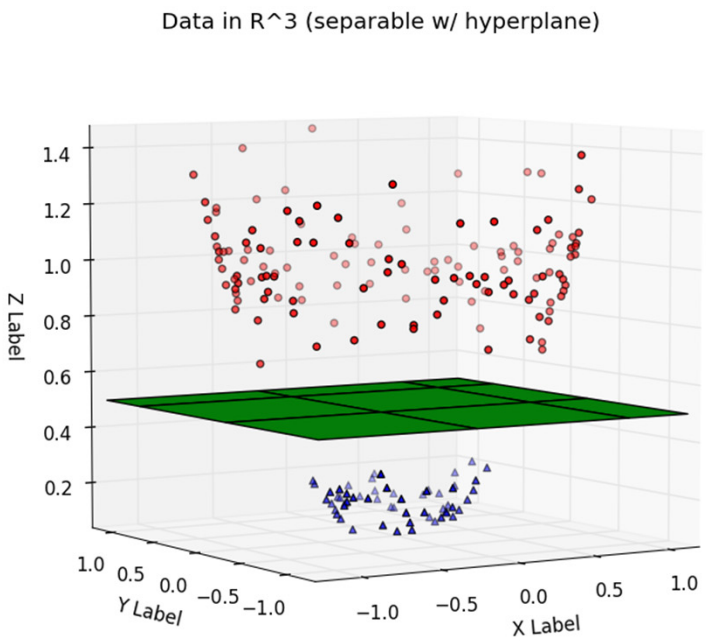




*[6] E. Kim*

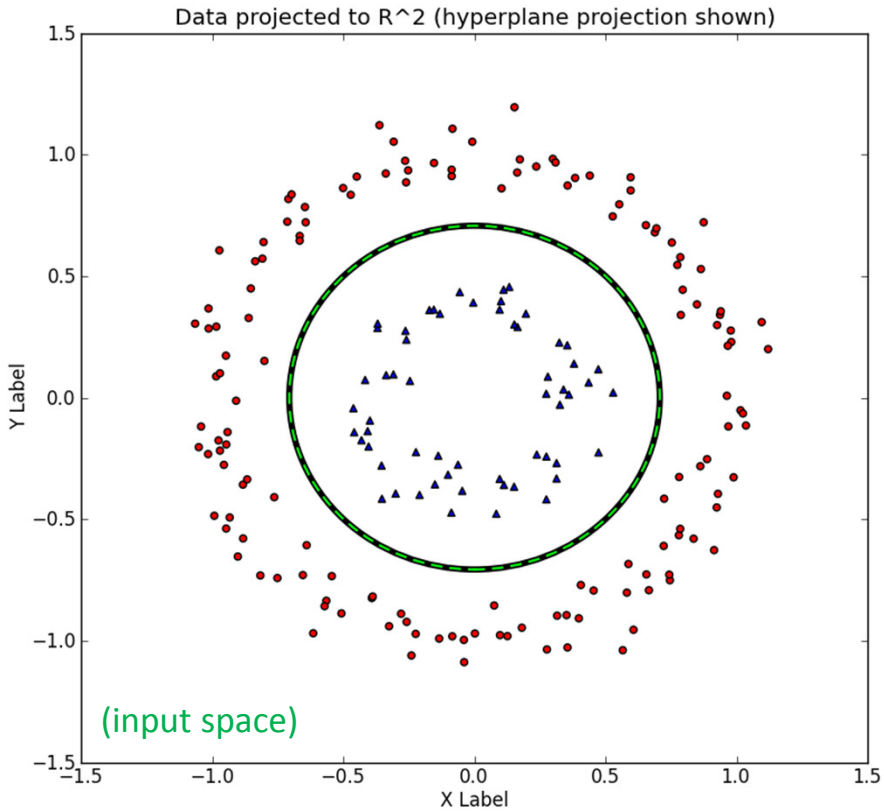
- **Problems: Not clear which type of mapping (search); optimization is computationally expensive task**

# Understanding Non-linear Transformations (3)

- Example: From 2 dimensional to 3 dimensional: $[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$
  - Separating hyperplane can be found and 'mapped back' to input space




(feature space)




*[6] E. Kim*

- **Problem: 'curse of dimensionality' – As dimensionality increases & volume of space too: sparse data!**

# Term Support Vector Machines – Revisited

- **Support Vector Machines (SVMs) are a classification technique developed ~1990**
- **SVMs perform well in many settings & are considered as one of the best <u>'out of the box classifiers'</u>**

***[1] An Introduction to Statistical Learning***

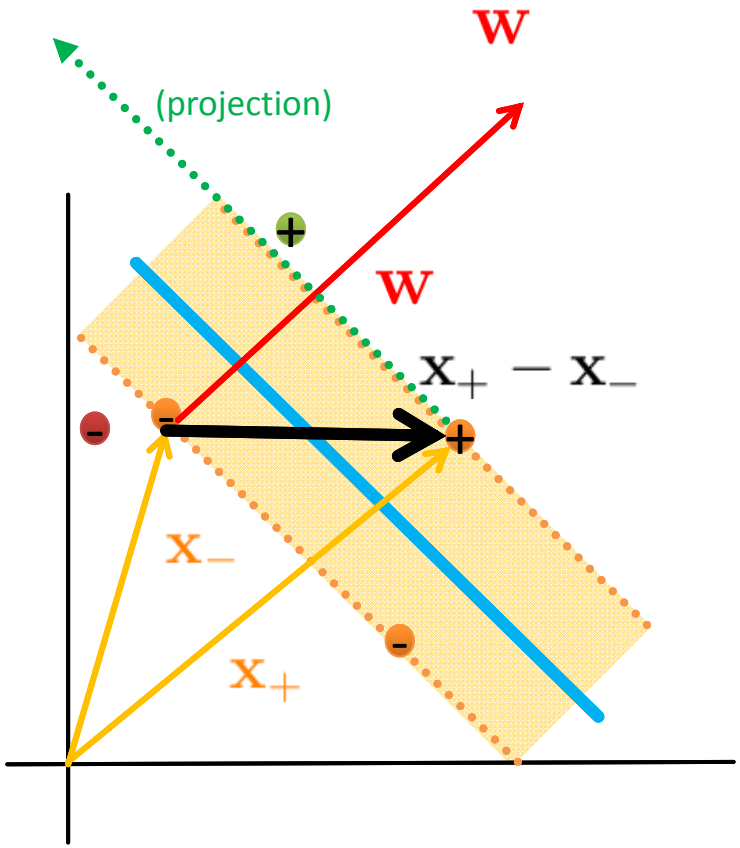
- Term detailed refinement into 'three separate techniques'
  - Practice: applications mostly use the SVMs with kernel methods
- 'Maximal margin classifier'
  - A simple and intuitive classifier with a 'best' linear class boundary
  - Requires that data is 'linearly separable'
- 'Support Vector Classifier'
  - Extension to the maximal margin classifier for non-linearly seperable data
  - Applied to a broader range of cases, idea of 'allowing some error'
- 'Support Vector Machines' → Using Non-Linear Kernel Methods
  - Extension of the support vector classifier
  - Enables non-linear class boundaries & via kernels;

# Constrained Optimization Steps SVM & Dot Product

- Rewrite formula: $\mathcal{L} = \frac{1}{2}\boxed{\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right) - \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)}$ (the same)

$$\cancel{- \sum \alpha_i y_i b} + \sum \alpha_i$$

(was 0)




(results in)

(optimization depends only on dot product of samples)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i \cdot \mathbf{x}_j} \quad (6)$$

- Equation to be solved by some quadratic programming package

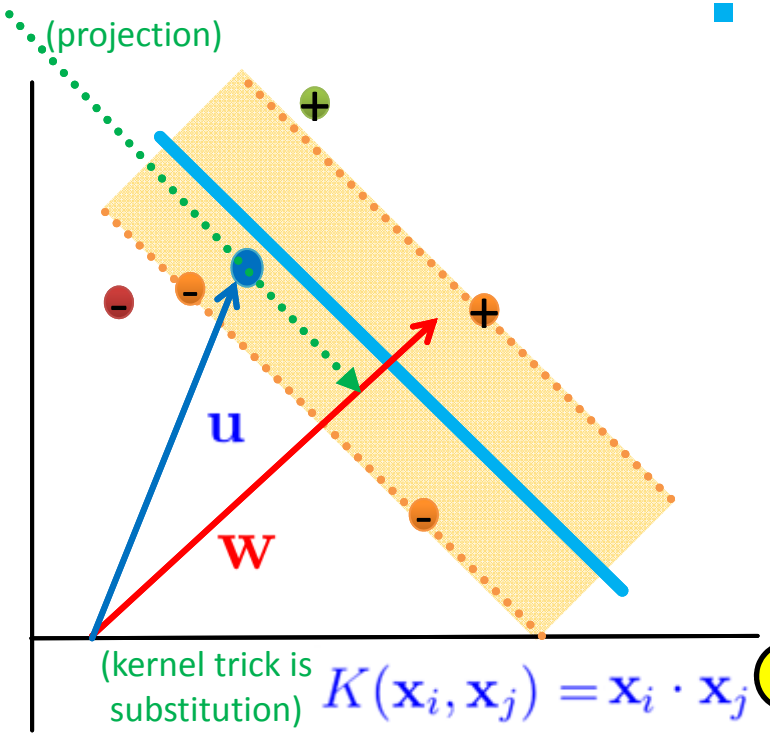
# Kernel Methods & Dot Product Dependency

- Use findings for decision rule

5 $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

1 $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ + ➡ $\sum \alpha_i y_i \boxed{\mathbf{x}_i \cdot \mathbf{u}_i} + b \geq 0$ +

(decision rule also depends on dotproduct)




- Dotproduct enables nice more elements
  - E.g. consider non linearly seperable data
  - Perform non-linear transformation $\Phi$ of the samples into another space (work on features)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i \cdot \mathbf{x}_j}$$ 6

(optimization depends only on dot product of samples)

➡ $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ (in optimization)

➡ $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{u}_i)$ (for decision rule above too)

(kernel trick is substitution) $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$

7 $\boxed{K(\mathbf{x}_i, \mathbf{x}_j)} = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

(trusted Kernel avoids to know Phi)

# Support Vector Machines & Kernel Methods

- **Support Vector Machines are extensions of the support vector classifier using kernel methods**
- **Support Vector Machines enable non-linear decision boundaries that can be efficiently computed**
- **Support Vector Machines avoids 'curse of dimensionality' and mapping search using a 'kernel trick'**

***[1] An Introduction to Statistical Learning***

- Non-linear transformations
  - Lead to high number of features → Computations become unmanageable
    (including the danger to run into 'curse of dimensionality')
- Benefits with SVMs
  - Enlarge feature space using 'kernel trick' → ensures efficient computations
  - Map training data into a higher-dimensional feature space using $\Phi$
  - Create a seperating 'hyperplane' with maximum margin in feature space
- Solve constraint optimization problem
  - Using Lagrange multipliers & quadratic programming (cf. earlier classifiers)
  - Solution involves the inner products of the data points (dot products)
  - Inner product of two r-vectors a and b is defined as $\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$
  - Inner product of two data points: $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$

# Linear SV Classifier Refined & Role of SVs

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

- Linear support vector classifier
  - Details w.r.t. inner products
  - With n parameters $\alpha_i, \quad i = 1, \ldots, n$ (Lagrange multipliers)
  - Use training set to estimate parameters
  - Estimate $\alpha_1, \ldots, \alpha_n$ and $\beta_0$ using $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

(between all pairs of training data points)

n (n – 1) / 2 number of pairs

- Evaluate $f(x)$ with a new point
  - Compute the inner product between new point x and each of the training points $x_i$

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

'big data' reduction & less computing

- Identify support vectors → Quadratic programming
  - $\alpha_i$ is zero most of the times (identified as not support vectors)
  - $\alpha_i$ is nonzero several times (identified as the support vectors)

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

(S with indices of support vectors)

***[1] An Introduction to Statistical Learning***

# The ('Trusted') Kernel Trick

- Summary for computation
  - All that is needed to compute coefficients are inner products
- Kernel Trick
  - Replace the inner product with a generalization of the inner product
  - K is some kernel function
- Kernel types
  - Linear kernel
  - Polynomial kernel

(sompute the hyperplane without explictly carrying out the map into the feature space)

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

(inner product used before)

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

$$K(x_i, x_{i'})$$

(kernel ~ distance measure)

(choosing a specific kernel type)

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$ (linear in features)

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$ (polynomial of degree d)

***[1] An Introduction to Statistical Learning***

- **Kernel trick refers to a mechanism of using different kernel functions (e.g. polynomial)**
- **A kernel is a function that quantifies the similarity of two data points (e.g. close to each other)**

# Kernel Trick – Example

- Consider again a simple two dimensional dataset
  - We found an ideal mapping $\Phi$ after long search
  - Then we need to transform the whole dataset according to $\Phi$

$$\Phi : (x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

  - Instead, with the 'kernel trick' we 'wait' and 'let the kernel do the job':

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \Rightarrow min_\alpha \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum \alpha_i$$

(no need to compute the mapping already)

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, 1)$$

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1$$

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^2 = K(\mathbf{u}, \mathbf{v})$$

(in transformed space still a dot product in the original space → no mapping needed)

(we can save computing time by do not perform the mapping)

- **Example shows that the dot product in the transformed space can be expressed in terms of a similarity function in the original space (here dot product is a similiarity between two vectors)**

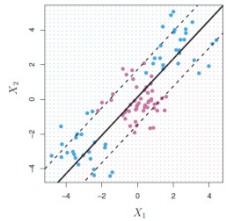
# Linear vs. Polynomial Kernel Example

- Linear kernel

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$ (linear in features)

  - Enables linear decision boundaries (i.e. like linear support vector classifier)

(observed useless for non-linear data)



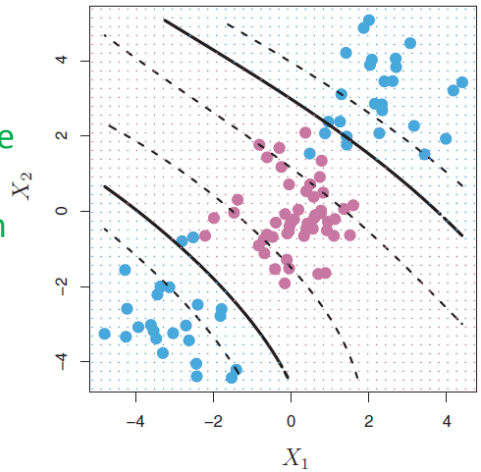
- Polynomial kernel

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$ (polynomial of degree d)

  - Satisfy Mercer's theorem = trusted kernel
  - Enables non-linear decision boundaries (when choosing degree d > 1)
  - Amounts to fit a support vector classifier in a higher-dimensional space
  - Using polynomials of degree d (d=1 linear support vector classifier)

(SVM with polynomial kernel of degree 3)

(significantly improved decision rule due to much more flexible decision boundary)

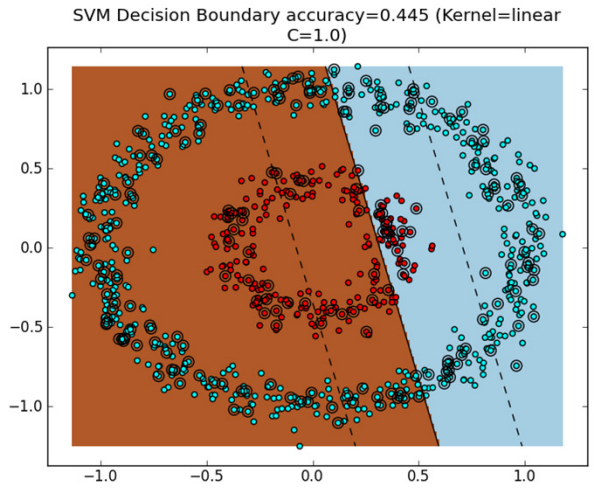



*[1] An Introduction to Statistical Learning*

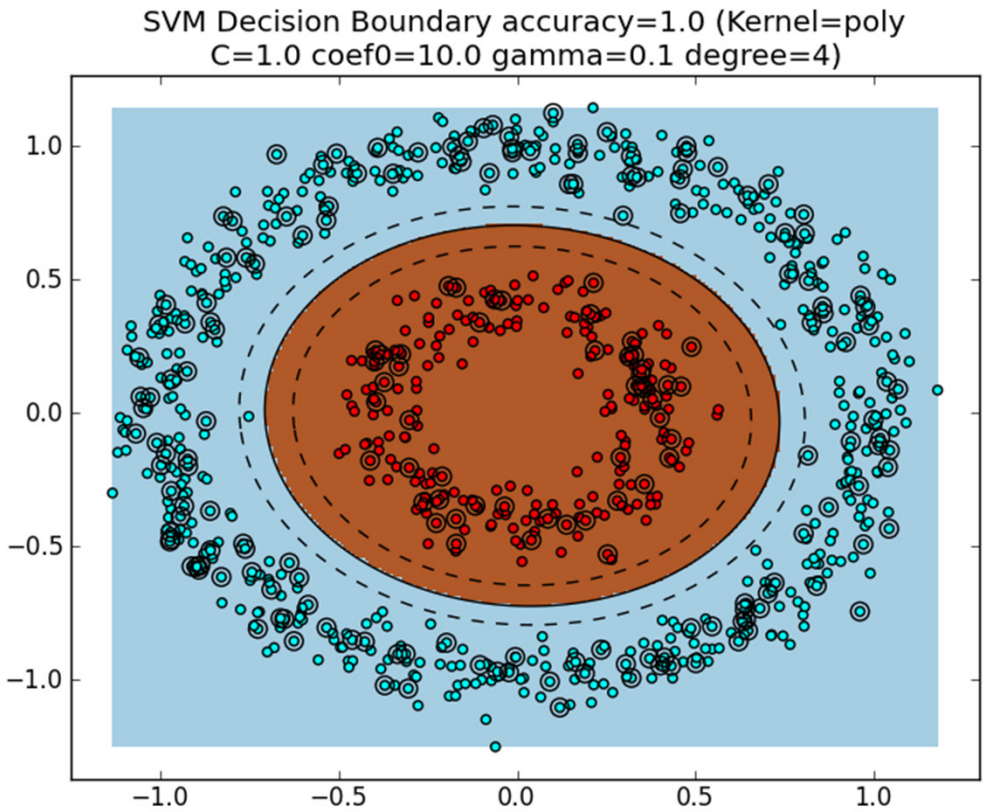
- **Polynomial kernel applied to non-linear data is an improvement over linear support vector classifiers**

# Polynomial Kernel Example

- Circled data points are from the test set




$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$




***[6] E. Kim***

# RBF Kernel

- Radial Basis Function (RBF) kernel (also known as radial kernel)
  - One of the mostly used kernel function
  - Uses parameter $\gamma$ as positive constant

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

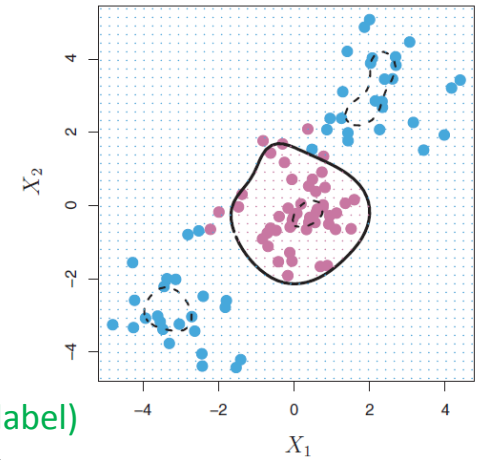
- 'Local Behaviour functionality'
  - Related to Euclidean distance measure (ruler distance) $d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
- Example
  - Use test data $x^* = (x_1^* \ldots x_p^*)^T$
  - Euclidean distance gives $x^*$ far from $x_i$

$\sum_{j=1}^{p} (x_j^* - x_{ij})^2$ (large value with large distance)

$K(x^*, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j^* - x_{ij})^2)$ (tiny value)

$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$ (training data $x_i$ plays no role for x* & its class label)

(SVM with radial kernel)



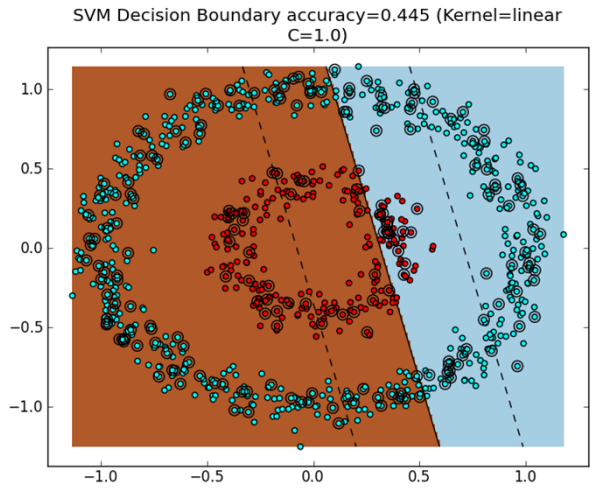

*[1] An Introduction to Statistical Learning*

- **RBF kernel have local behaviour (only nearby training data points have an effect on the class label**

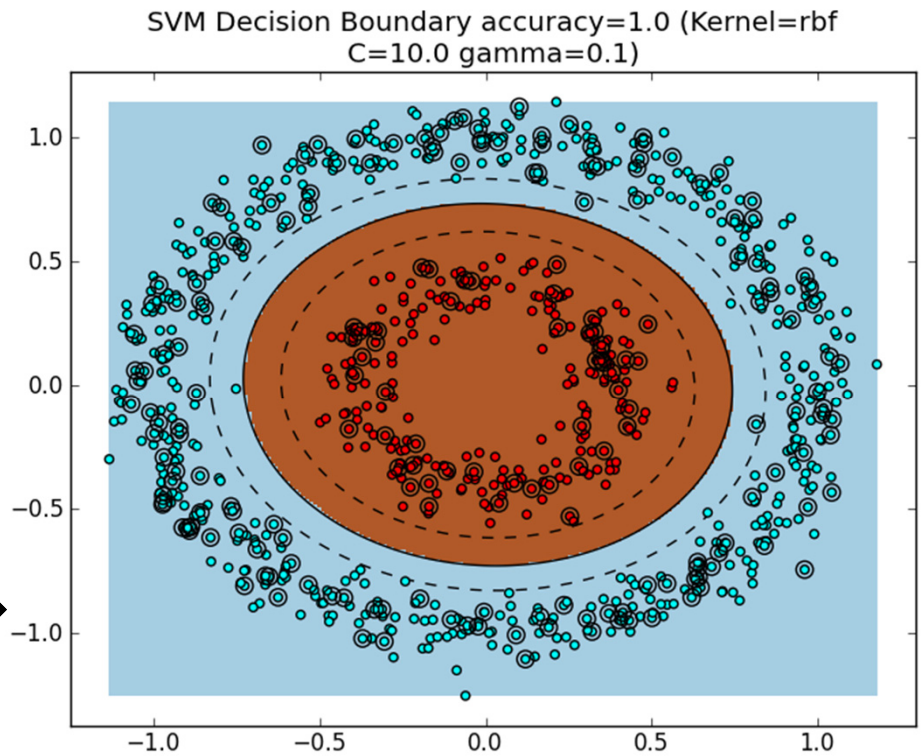
# RBF Kernel Example

- Circled data points are from the test set

(similiar decision boundary as polynomial kernel)




$$K(x^*, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j^* - x_{ij})^2)$$




***[6] E. Kim***

# Exact SVM Definition using non-linear Kernels

- **True Support Vector Machines are Support Vector Classifiers combined with a non-linear kernel**
- **There are many non-linear kernels, but mostly known are polynomial and RBF kernels**

***[1] An Introduction to Statistical Learning***

- General form of SVM classifier
  - Assuming non-linear kernel function K
  - Based on 'smaller' collection S of SVs

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$$

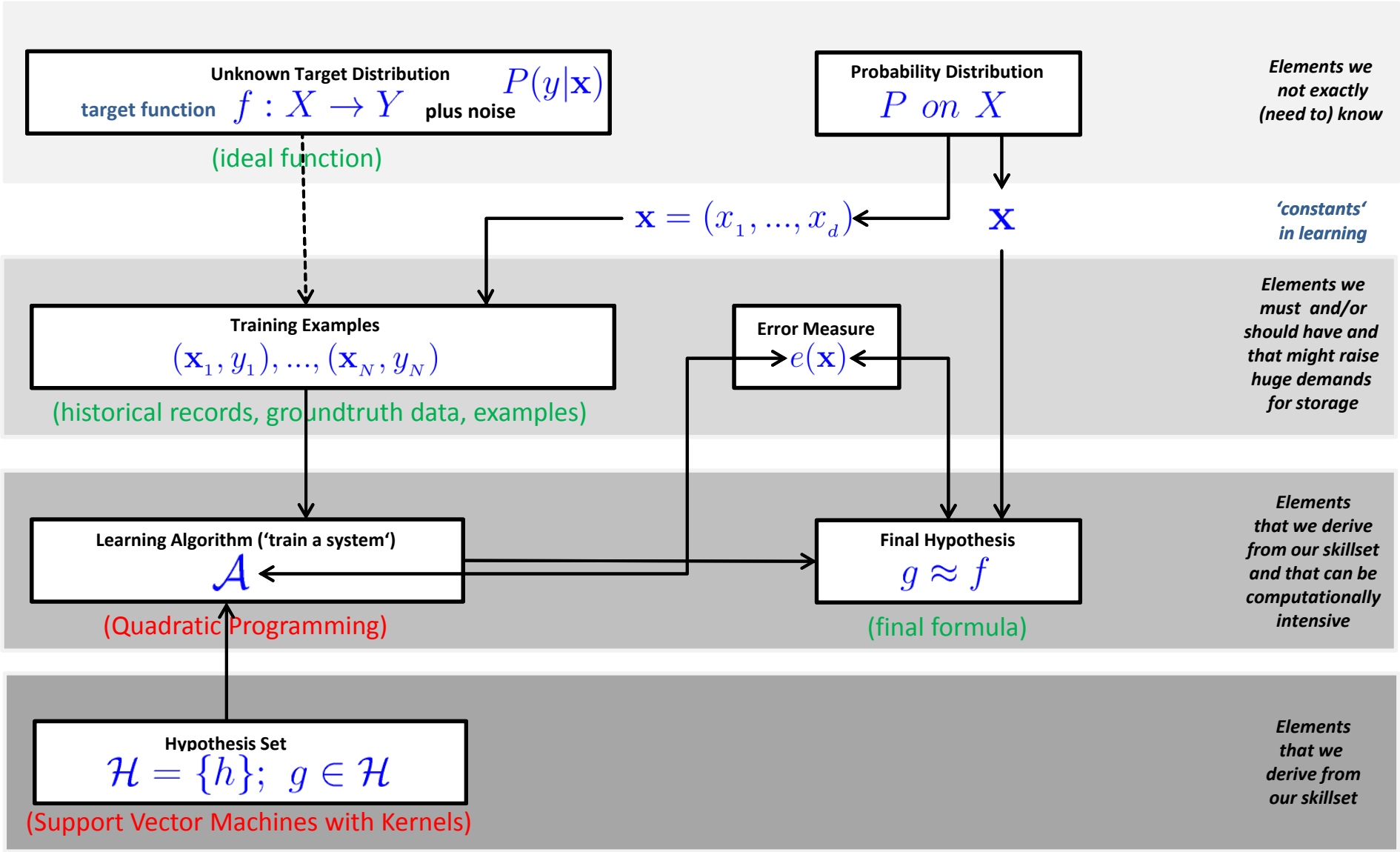
- Major benefit of Kernels: Computing done in original space

(independent from transformed space)

- Linear Kernel $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$ (linear in features)
- Polynomial Kernel $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$ (polynomial of degree d)
- RBF Kernel $K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$ (large distance, small impact)

(the win: kernel can compute this without ever computing the coordinates of the data in that space, next slides)

# Solution Tools: Support Vector Classifier & QP Algorithm



# Non-Linear Transformations with Support Vector Machines

- Same idea: work in z space instead of x space with SVMs
  - Understanding effect on solving (labels remain same)
  - SVMs will give the 'best seperator'

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \boxed{\mathbf{x}_n^T \mathbf{x}_m}$$

(replace this simply with z's obtained by $\Phi$ )

(result from this new inner product is given to quadratic programming optimization as input as before )

  - Value: inner product is done with z instead of x – the only change
  - Result after quadratic programming is hyperplane in z space using the value
- Impacts of $\Phi$ to optimization
  - From linear to 2D → probably no drastic change
  - From 2D to million-D → sounds like a drastic change but just inner product
  - Input for $K(x_i, x_i')$ remains the number of data points (nothing to do with million-D)
  - Computing longer million-D vectors is 'easy' – optimization steps 'difficult'
- **Infinite-D Z spaces are possible since the non-linear transformation does not affect the optimization**

# Kernels & Infinite Z spaces

$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \emptyset(\boldsymbol{x}_i)^T \emptyset(\boldsymbol{x}_j)$

(maps data to higher-dimensional feature spaces)

- Understanding advantage of using a kernel
  - Better than simply enlarging the feature space
  - E.g. using functions of the original features like $X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$
- Computational advantages
  - By using kernels only compute $K(x_i, x_i')$
  - Limited to just all $\binom{n}{2}$ distinct pairs $i,\ i'$

    (number of 2 element sets from n element set)
  - Computing without explicitly working in the enlarged feature space
  - Important because in many applications the enlarged feature space is large

    (computing would be infeasible then w/o kernels)
- Infinite-D Z spaces $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \emptyset(\boldsymbol{x}_i)^T \emptyset(\boldsymbol{x}_j)$
  - Possible since all that is needed to compute coefficients are inner products

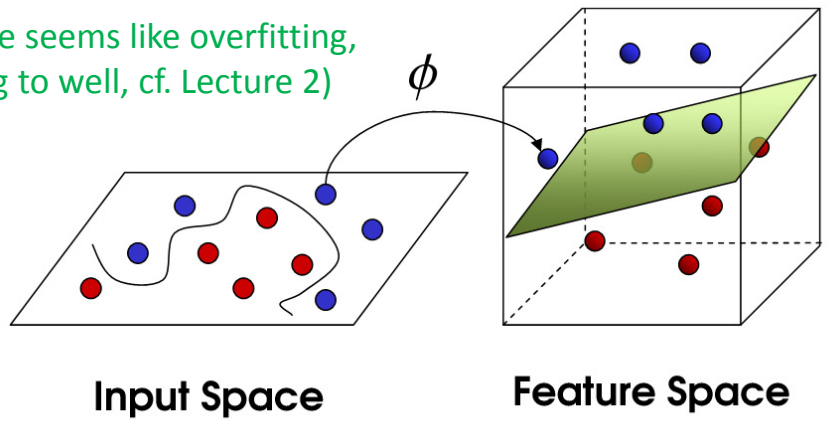
***[1] An Introduction to Statistical Learning***

- **Kernel methods like RBF have an implicit and infinite-dimensional features space that is not 'visited'**

# Visualization of SVs

- Problem: z-Space is infinite (unknown)
  - How can the Support Vectors (from existing points) be visualized?
  - Solution: non-zero alphas have been the identified support vectors
    (solution of quadratic programming optimization will be a set of alphas we can visualize)
  - Support vectors exist in Z – space (just transformed original data points)
  - Example: million-D means a million-D vector for $\mathbf{w}$
  - But number of support vector is very low, expected $E_{out}$ is related to #SVs
    (generalization behaviour despite million-D & snake-like overfitting)

(snake seems like overfitting, fitting to well, cf. Lecture 2)




*[7] Visualization of high-dimensional space*

- **Counting the number of support vectors remains to be a good indicator for generalization behaviour even when performing non-linear transforms and kernel methods that can lead to infinite-D spaces**

(rule of thumb)

# Parallel & Scalable PiSVM - Parameters

```
[train001@j3l02 pisvm-1.2.1]$ ./pisvm-train
Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
        0 -- C-SVC
        1 -- nu-SVC
        2 -- one-class SVM
        3 -- epsilon-SVR
        4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/k)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 40)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight: set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-o n: max. size of working set
-q n: max. number of new variables entering working set
flags:
-D: Assume the feature vectors are dense (default: sparse)
```

# Training Rome on JUROPA3 – Job Script (RBF)

- Use Rome Dataset with paralle & scalable piSVM tool
  - Parameters are equal to the serial libsvm and some additional

```
#!/bin/bash -x
#SBATCH--nodes=2
#SBATCH--ntasks=48
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=01:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=train-rome-rbf-2-48-24

### location executable
PISVM=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-train

### location data
TRAINDATA=/homea/hpclab/train001/data/rome/sdap_area_all_training.el

### submit
srun $PISVM -D -o 1024 -q 512 -c 100 -g 8 -t 2 -m 1024 -s 0 $TRAINDATA
```

- **Note the tutorial reservation with –reservation=ml-hpc-2 just valid for today on JUROPA3**

# Testing Rome on JUROPA3 – Job Script (RBF)

- Use Rome Dataset with paralle & scalable piSVM tool
  - Parameters are equal to the serial libsvm and some additional parameters for paralellization

```
#SBATCH--ntasks-per-node=24
#SBATCH--output=mpi-out.%j
#SBATCH--error=mpi-err.%j
#SBATCH--time=01:00:00
#SBATCH--partition=batch
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=pred-rome-2-48-24

### location executable
PISVMPRED=/homea/hpclab/train001/tools/pisvm-1.2.1/pisvm-predict

### location data
TESTDATA=/homea/hpclab/train001/data/rome/sdap_area_all_test.el

### trained model data
MODELDATA=/homea/hpclab/train001/tools/pisvm-1.2.1/sdap_area_all_training.el.model

### submit
srun $PISVMPRED $TESTDATA $MODELDATA results.txt
```

- **Note the tutorial reservation with –reservation=ml-hpc-2 just valid for today on JUROPA3**

# Testing Rome on JUROPA3 – Check Outcome

- The output of the training run is a model file
  - Used for input for the testing/predicting phase
  - In-sample property → Support vectors of the model

```
### trained model data
MODELDATA=/homea/hpclab/train001/tools/pisvm-1.2.1/sdap_area_all_training.el.model
```

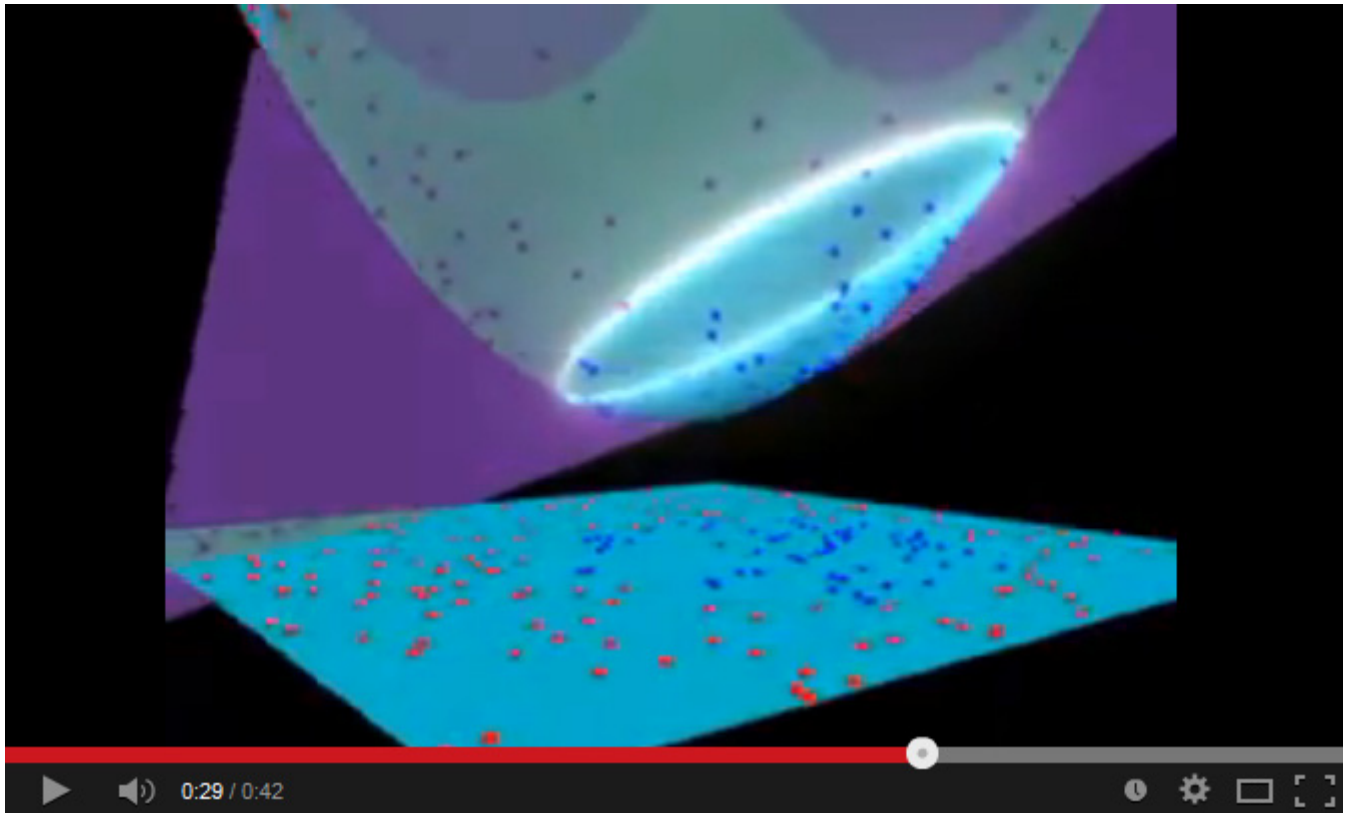
- The output of the testing/predicting phase is accuracy
  - Accuracy measurement of model performance (cf. Lecture 1)

```
[train001@j3l02 pisvm-1.2.1]$ more mpi-out.15244544
Accuracy = 98.0558% (684291/697859) (classification)
Mean squared error = 0.195456 (regression)
Squared correlation coefficient = 0.968742 (regression)
```

- Output of linear SVM was as follows:

```
[train001@j3l02 pisvm-1.2.1]$ more mpi-out.15244542
Accuracy = 91.5994% (639235/697859) (classification)
Mean squared error = 1.04794 (regression)
Squared correlation coefficient = 0.835385 (regression)
```

# [Video] SVM with Polynomial Kernel Example




*[8] YouTube, SVM with Polynomial Kernel*

# Lecture Bibliography

# Lecture Bibliography


- [1] An Introduction to Statistical Learning with Applications in R,
  Online: http://www-bcf.usc.edu/~gareth/ISL/index.html
- [2] Big Data Tips, 'SVM-Train'
  Online: http://www.big-data.tips/svm-train
- [3] Rome Dataset, B2SHARE,
  Online: http://hdl.handle.net/11304/4615928c-e1a5-11e3-8cd7-14feb57d12b9
- [4] YouTube Video, 'Text Classification 2: Maximum Margin Hyperplane',
  Online: https://www.youtube.com/watch?v=dQ68FW7p97A
- [5] YouTube Video, 'Cascade SVM',
  Online:
- [6] E. Kim, 'Everything You Wanted to Know about the Kernel Trick',
  Online: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html
- [7] Visualization of high-dimensional space,
  Online: http://i.imgur.com/WuxyO.png
- [8] YouTube, 'SVM with Polynomial Kernel',
  Online: https://www.youtube.com/watch?v=3liCbRZPrZA



- Acknowledgements and more Information: Yaser Abu-Mostafa, Caltech Lecture series, YouTube

# Slides Available at http://www.morrisriedel.de/talks