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

March 7th, 2018
JSC, Germany
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 'out of the box classifiers'

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 separable 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;

[1] An Introduction to Statistical Learning
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)

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

(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)
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 decision rule:
    \( \mathbf{w} \cdot \mathbf{u} + b \geq 0 \rightarrow \text{means } + \)
    (given that \( b \) and \( \mathbf{w} \) are unknown to us)
    (constraints are not enough to fix particular \( b \) or \( \mathbf{w} \), need more constraints to calculate \( b \) or \( \mathbf{w} \))
Geometric SVM Interpretation and Setup (2)

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

- 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)

    \[
    y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0
    \]
Determine the ‘width of the margin’
- Difference between positive and negative SVs: $x_+ - x_-$
- Projection of $x_+ - x_-$ onto the vector $w$
- The vector $w$ is a normal vector, magnitude is $\|w\|$

Unit vector is helpful for ‘margin width’
- Projection (dot product) for margin width:
  $$\frac{w}{\|w\|}$$
  (unit vector)
- When enforce constraint:
  1. $y_i(x_i \cdot w + b) - 1 = 0$
  2. $y_i = +$
  3. $y_i = -$
Constrained Optimization Steps SVM (1)

- Use ‘constraint optimization’ of mathematical toolkit

- Idea is to ‘maximize the width’ of the margin: \( \max \frac{2}{\|w\|} \) (drop the constant 2 is possible here)

- Next: Find the extreme values
  - Subject to constraints
    \[ y_i(x_i \cdot w + b) - 1 = 0 \]
Lecture 7 – Support Vector Machines and Kernel Methods

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

2. \[ y_i(x_i \cdot w + b - 1) = 0 \]

- Introduce a multiplier for each constraint
  \[ \mathcal{L}(\alpha) = \frac{1}{2}\|w\|^2 - \sum \alpha_i [y_i(x_i \cdot w + b) - 1] \]
  (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. \( w \)
  - Second differentiate w.r.t. \( b \)
  (derivative gives the gradient, setting 0 means extremum like min)
Constrained Optimization Steps SVM (3)

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

- First differentiate w.r.t. \( w \)
  \[ \frac{\partial \mathcal{L}}{\partial w} = w - \sum \alpha_i y_i x_i = 0 \]
  (derivative gives the gradient, setting 0 means extremum like min)

- Simple transformation brings:
  \[ w = \sum \alpha_i y_i x_i \] (i.e. vector is linear sum of samples)
  (recall: non zero for support vectors, rest zero \( \rightarrow \) even less samples)

- Second differentiate w.r.t. \( b \)
  \[ \frac{\partial \mathcal{L}}{\partial b} = - \sum \alpha_i y_i = 0 \]
  \[ \sum \alpha_i y_i = 0 \]
Constrained Optimization Steps SVM (4)

- Lagrange gives: \( \mathcal{L}(\alpha) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (x_i \cdot w + b) - 1] \)
- Find minimum
  - Quadratic optimization problem
  - Take advantage of \( w = \sum \alpha_i y_i x_i \)
  \[ \mathcal{L} = \frac{1}{2} \left( \sum \alpha_i y_i x_i \right) \cdot \left( \sum \alpha_j y_j x_j \right) - \sum \alpha_i y_i x_i \cdot \left( \sum \alpha_j y_j x_j \right) - \sum \alpha_i y_i b + \sum \alpha_i \]
  \( \sum \alpha_i y_i = 0 \)
Constrained Optimization Steps SVM (5)

- Rewrite formula: \( \mathcal{L} = \frac{1}{2} \left( \sum \alpha_i y_i x_i \right) \cdot \left( \sum \alpha_j y_j x_j \right) - \sum \alpha_i y_i x_i \cdot \left( \sum \alpha_j y_j x_j \right) \)

- Equation to be solved by some quadratic programming package

- Constrained optimization depends only on dot product of samples
Use of SVM Classifier to Perform Classification

- Use findings for decision rule

\[ w = \sum \alpha_i y_i x_i \]

1. \( w \cdot u + b \geq 0 \) 

\[ \sum \alpha_i y_i x_i \cdot u_i + b \geq 0 \] (decision rule also depends on dotproduct)
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 separable data
  - Binary classification problem (two class classification)
  - Test set: a vector $x^*$ with test observations

$X_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$

$(n \times p$-dimensional vectors)$

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

(class labels) (two classes)

$x^* = \begin{pmatrix} x_{1}^* \\ \vdots \\ x_{p}^* \end{pmatrix}^T$

Maximal Margin Classifiers create a separating hyperplane that separates 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 separating 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 separating hyperplane’

- The maximal margin hyperplane is the separating 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
  \[ \text{sign of } f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^* \]
  (\(\beta_s\) 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

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 \( \beta \) coefficients)

\[
\begin{align*}
\text{maximize } & M \\
\text{subject to } & \sum_{j=1}^{p} \beta_j^2 = 1 \\
\end{align*}
\]

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

- 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 \( b \))

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

\[
\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m
\]

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

- 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

- Use of already established Lagrangian method:

\[
\min_{w,b} \left\{ \frac{1}{2} \| w \|^2 \right\}
\text{subject to } y_i (w \cdot x_i - b) \geq 1
\]
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)**

- **N x N, usually not sparse**
- **Computational complexity** relies in the following:
  \[
  L(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m
  \]
  
  \[
  \begin{bmatrix}
  y_1 y_1 x_1^T x_1 & y_1 y_2 x_1^T x_2 & \cdots & y_1 y_N x_1^T x_N \\
  \vdots & \vdots & \ddots & \vdots \\
  y_N y_1 x_N^T x_1 & y_N y_2 x_N^T x_2 & \cdots & y_N y_N x_N^T x_N 
  \end{bmatrix}
  \]
  (quadratic coefficients, alphas are result from QP)

- **Generalization measure**: #SVs as ‘in-sample quantity’ → 10SVs/1000 samples ok, 500SVs/1000 bad
- **Reasoning towards overfitting due to a large number of SVs** (fit many, small margin, gives bad $E_{out}$)
Solution Tools: Maximal Margin Classifier & QP Algorithm

**Unknown Target Distribution**
\[ f : X \rightarrow Y \]
\[ P(y|x) \]

(ideal function)

**Probability Distribution**
\[ P \text{ on } X \]

Elements we not exactly (need to) know

**Training Examples**
\[ (x_1, y_1), \ldots, (x_N, y_N) \]

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

**Error Measure**
\[ e(x) \]

Elements that we derive from our skillset and that can be computationally intensive

**Learning Algorithm** ('train a system')
\[ \mathcal{A} \]

(Quadratic Programming)

Elements that we derive from our skillset

**Final Hypothesis**
\[ g \approx f \]

(final formula)

**Hypothesis Set**
\[ \mathcal{H} = \{ h \} ; \ g \in \mathcal{H} \]

(Maximal Margin Classifier)
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 separable data

**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*)

*(modified from [1] An Introduction to Statistical Learning)*
Exercises – Submit piSVM & Rome (linear)
JURECA System – SSH Login

- Use your account train004 - train050
- Windows: use putty / MobaXterm
- UNIX: ssh trainXYZ@jureca.fz-juelich.de
- Example

```
adminuser@linux-Bdjg:--$ ssh train001@jureca.fz-juelich.de
Warning: the ECDSA host key for 'jureca.fz-juelich.de' differs from the key for the IP address '134.94.33.9'
Offending key for IP in /home/adminuser/.ssh/known_hosts:12
Matching host key in /home/adminuser/.ssh/known_hosts:19
Are you sure you want to continue connecting (yes/no)? yes
Last login: Mon Aug 21 14:29:03 2017 from zam2036.zam.kfa-juelich.de
************************************************************************************
* Welcome to JURECA
* *
* Information about the system, latest changes, user documentation and FAQs:
*  http://www.fz-juelich.de/ias/jsc/jureca
************************************************************************************
* ### Known Issues ###
* *
* An up-to-date list of known issues on the system is maintained at
*  http://www.fz-juelich.de/ias/jsc/jureca-known-issues
* Open issues:
* - Intel compiler error with std::valarray and
*  optimized headers, added 2016-03-20
```

➤ Remember to use your own trainXYZ account in order to login to the JURECA system
Rome Remote Sensing Dataset (cf. Lecture 6)

- Data is already available in the tutorial directory

```
[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 .
-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 12645592 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.:
    - Module load Intel
    - Module load IntelMPI
Parallel & Scalable PiSVM – Parameters

- 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

```
=train001@j3l02 pisvm-1.2.1]$ ./pisvm-train
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 cache_size : 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 JURECA – Job Script (linear)

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

```bash
#!/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 JURECA
Testing Rome on JURECA – Job Script (linear)

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

    ```bash
    #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 JURECA
Testing Rome on JURECA – 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

```plaintext
### 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)
```
Large Margin Classifiers

- Learn a decision boundary $\mathbf{w}$: $\mathbf{d}^T \mathbf{w} > 0$ iff $\mathbf{d}$ is positive
- Problem: many such $\mathbf{w}$ (assuming examples separable)
- Maximum-margin: “buffer zone” around boundary
  - as far as possible from nearest training examples: $\mathbf{d}^T \mathbf{w} > \theta$ (+)
- Support Vector Machine (SVM)
  - best classification accuracy
  - can be slow to train (use SVM\textsuperscript{light})
- Passive Aggressive (PA)
  - fast to train, streaming
  - accuracy can be lower
- What works in practice:
  - don’t use non-linear versions
  - don’t do feature selection (LSI)

[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 ‘out of the box classifiers’

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)

**Approach**
- Generalization of the ‘maximal margin classifier’
- Include non-separable cases with a soft-margin (almost instead of exact)
- Being more robust w.r.t. extreme values (e.g. outliers) allowing small errors

(outlier) (significant reduction of the maximal margin)

(get most & but not all training data correctly classified)

(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)

[1] An Introduction to Statistical Learning
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)

[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 violation of margins
  - Adding slack variables $\epsilon_1, \ldots, \epsilon_n$

$$\begin{align*}
\max_{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_n} & \quad M \\
\text{subject to} & \quad \sum_{j=1}^{p} \beta_j^2 = 1, \\
& \quad y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \text{(slightly violate the Margin)}
\end{align*}$$

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

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

(C is used here to bound the error)

- $C$ 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

[C] An Introduction to Statistical Learning

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

Lecture 7 – Support Vector Machines and Kernel Methods
Solution Tools: Support Vector Classifier & QP Algorithm

- **Unknown Target Distribution**
  - target function \( f : X \rightarrow Y \) plus noise
  - \( P(y|x) \)
  - (ideal function)

- **Probability Distribution**
  - \( P \text{ on } X \)

- **Training Examples**
  - \( (x_1, y_1), \ldots, (x_N, y_N) \)
  - (historical records, groundtruth data, examples)

- **Error Measure**
  - \( e(x) \)

- **Learning Algorithm (‘train a system’)**
  - \( \mathcal{A} \)
  - (Quadratic Programming)

- **Final Hypothesis**
  - \( g \approx f \)
  - (final formula)

- **Hypothesis Set**
  - \( \mathcal{H} = \{h\}; \ g \in \mathcal{H} \)
  - (Support Vector Classifier)

**Elements**
- We not exactly (need to) know
- ‘constants’ in learning
- We must and/or should have and that might raise huge demands for storage
- Elements that we derive from our skillset and that can be computationally intensive
- Elements that we derive from our skillset
Support Vector Classifier – Solving & Limitations

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

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 separable data

modified from [1] An Introduction to Statistical Learning

... but maybe there are more advanced techniques that help...
[Video] Training Process of Support Vector Machines

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

\[ X_1, X_2, \ldots, X_p \] (previously used \( p \) features)
\[ X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2 \] (new \( 2p \) features)

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

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

[1] An Introduction to Statistical Learning
Understanding Non-Linear Transformations (1)

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

(still linear models applicable)

(named as x-space)
(also called input space)

(‘changing constants’)

(named as z-space)
(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


- 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

- 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 ‘out of the box classifiers’

- 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 separable 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:
  \[ \mathcal{L} = \frac{1}{2} \left( \sum \alpha_i y_i x_i \right) \cdot \left( \sum \alpha_j y_j x_j \right) - \sum \alpha_i y_i x_i \cdot \left( \sum \alpha_j y_j x_j \right) - \sum \alpha_i y_i b + \sum \alpha_i \]

  (the same)
  (was 0)
  (optimization depends only on dot product of samples)

  \[ \mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j \]

  (results in)

- Equation to be solved by some quadratic programming package
Kernel Methods & Dot Product Dependency

- Use findings for decision rule

\[ w = \sum \alpha_i y_i x_i \]

\[ w \cdot u + b \geq 0 \]

\[ \sum \alpha_i y_i x_i \cdot u + b \geq 0 \]

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

\[ L = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j) \]

\( \Phi(x_i) \cdot \Phi(x_j) \) (in optimization)

\( \Phi(x_i) \cdot \Phi(u_j) \) (for decision rule above too)

\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]

\( K(x_i, x_j) = x_i \cdot x_j \)

Lecture 7 – Support Vector Machines and Kernel Methods
Support Vector Machines are extensions of the support vector classifier using kernel methods.
- Non-linear transformations
  - Lead to high number of features \( \rightarrow \) Computations become unmanageable
- Benefits with SVMs
  - Enlarge feature space using ‘kernel trick’ \( \rightarrow \) ensures efficient computations
  - Map training data into a higher-dimensional feature space using \( \Phi \)
  - Create a separating ‘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
  \[ \langle a, b \rangle = \sum_{i=1}^{T} a_i b_i \]
  - Inner product of two data points: \[ \langle x_i, x_{i'} \rangle = \sum_{j=1}^{T} x_{ij} x_{i'j} \]

[1] An Introduction to Statistical Learning

(including the danger to run into ‘curse of dimensionality’)
Linear SV Classifier Refined & Role of SVs

- Linear support vector classifier
  - Details w.r.t. inner products
  - With n parameters $\alpha_i$, $i = 1, \ldots, n$ (Lagrange multipliers)
  - Use training set to estimate parameters
  - Estimate $\alpha_1, \ldots, \alpha_n$ and $\beta_0$ using $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$
    - $n (n - 1) / 2$ number of pairs
  - Evaluate $f(x)$ with a new point
    - Compute the inner product between new point $x$ and each of the training points $x_i$
  - Identify support vectors $\Rightarrow$ 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)

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

\[ \langle x_i, x_i' \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \]
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
  - $K(x_i, x_{i'})$
  - $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$ (linear in features)
  - $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$ (polynomial of degree $d$)

- **Kernel types**
  - **Linear kernel**
  - **Polynomial kernel**

- **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)

- **Equation**
  - $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$
  - (compute the hyperplane without explicitly carrying out the map into the feature space)
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’:

\[
K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)
\]

\[
\min_{\alpha} \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum \alpha_i
\]

(no need to compute the mapping already)

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

\[
\Phi(u) \cdot \Phi(v) = u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1
\]

\[
\Phi(u) \cdot \Phi(v) = (u \cdot v + 1)^2 = K(u, v)
\]

(in transformed space still a dot product in the original space \(\rightarrow\) 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 similarity between two vectors)
Linear vs. Polynomial Kernel Example

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

- **Polynomial kernel**
  - 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)

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

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

(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

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

RBF Kernel

- **Radial Basis Function (RBF) kernel**
  - One of the mostly used kernel function
  - Uses parameter $\gamma$ as positive constant
  - ‘Local Behaviour functionality’
    - Related to **Euclidean distance** measure (ruler distance)

- **Example**
  - Use test data $x^* = (x^*_1 \ldots x^*_p)^T$
  - Euclidean distance gives $x^*$ far from $x_i$

\[
K(x^*, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x^*_j - x^*_{ij})^2)
\]

\[
d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
\]

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

[1] An Introduction to Statistical Learning

Lecture 7 – Support Vector Machines and Kernel Methods
RBF Kernel Example

- Circled data points are from the test set

(similar decision boundary as polynomial kernel)

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

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

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

- 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

Unknown Target Distribution

\[ f : X \rightarrow Y \quad P(y|x) \] (ideal function)

Elements we not exactly (need to) know

Training Examples

\((x_1, y_1), \ldots, (x_N, y_N)\)

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

Learning Algorithm ('train a system')

\[ A \]

(Quadratic Programming)

Elements that we derive from our skillset and that can be computationally intensive

Final Hypothesis

\[ g \approx f \]

(final formula)

Elements that we derive from our skillset

Error Measure

\[ e(x) \]

Probability Distribution

\[ P \text{ on } X \]

Elements 'constants' in learning

Hypothesis Set

\[ \mathcal{H} = \{h\}; \quad g \in \mathcal{H} \]

(Support Vector Machines with Kernels)
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 separator’
  
  \[ \mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m \]  
  (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 \( \rightarrow \) probably no drastic change
  - From 2D to million-D \( \rightarrow \) 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

- **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**
  - $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$
  - Possible since all that is needed to compute coefficients are inner products

---

[1] An Introduction to Statistical Learning
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)

- 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@j302 psvm-1.2.1]$ ./pisvm-train
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)
-f 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 cache-size : 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 JURECA – Job Script (RBF)

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

```bash
#!/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 JURECA
Testing Rome on JURECA – Job Script (RBF)

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

```bash
#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 JURECA
Testing Rome on JURECA – 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

```bash
### trained model data
MODELDATA=/homea/hpclab/train001/tools/pismv-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@j3lo2 psvm-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@j3lo2 psvm-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
  Online: http://www.big-data.tips/svm-train
- [3] Rome Dataset, B2SHARE,
  Online: http://hdl.handle.net/11304/4615928c-e1a5-11e3-8cd7-14feb57d12b9
  Online: https://www.youtube.com/watch?v=dQ68FW7p97A
  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
  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