# Simons' SVM: a new SVM library

Ingo Steinwart
University of Stuttgart

January 29, 2016

# Overview

- Philosophy of the library
- The user view
- Implementational aspects
- Some comparisons

# Philosophy of the library

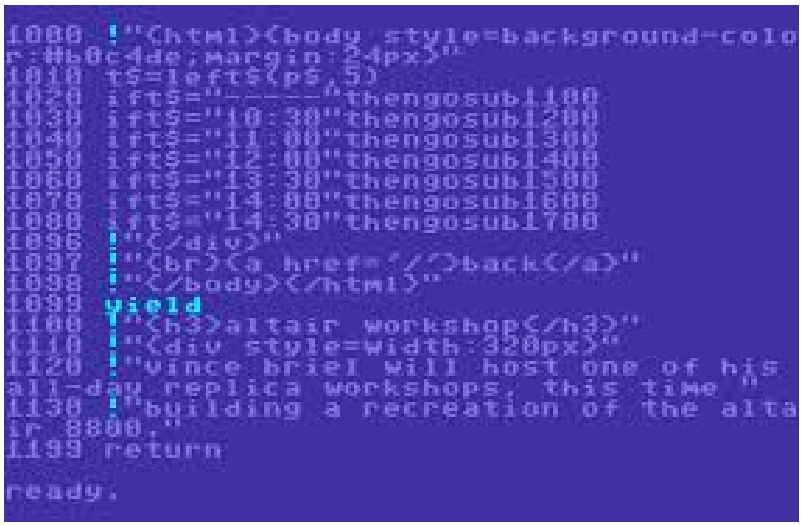
# What a Stupid Name?!

## My very first computer



# What a Stupid Name?!

## And its interface



# What a Stupid Name?!

## And its interface

```
1000 !"<html><body style=background-colo
r:#b0c4de;margin:24px>"
1010 t$=left$(p$,5)
1020 if t$="-----" then gosub1100
1030 if t$="10:30" then gosub1200
1040 if t$="11:00" then gosub1300
1050 if t$="12:00" then gosub1400
1060 if t$="13:30" then gosub1500
1070 if t$="14:00" then gosub1600
1080 if t$="14:30" then gosub1700
1096 !"</div>"
1097 !"<br><a href='/'>back</a>"
1098 !"</body></html>"
1099 yield
1100 !"<h3>altair workshop</h3>"
1110 !"<div style=width:320px>"
1120 !"vince briel will host one of his
all-day replica workshops, this time "
1130 !"building a recreation of the alta
ir 8800."
1199 return

ready.
```



# What a Stupid Name?!

## An Alternative






# Major Aims

- Speed
- Fully automated learning
- Flexibility

# Speed

**Speed is achieved by the following approaches**

- Re-use of kernel matrix
- A new solver design
- Threading on physical cores
- SIMD instructions (vectorization)
- Compilation with target specific options

# Fully Automated Learning

**The user should only set parameters he/she can understand**

- Built-in cross-validation
- Default values are in many cases good
- User-control over computational resources
- User-control over learning scenario
- Three interfaces: turn-key, advanced user, and R

# Flexibility

**Flexibility is achieved by several means**

- Terminal/file-based interface allows for wrappers in high-level languages (R, Matlab, . . . )
- 3-step approach (training, paramater selection, and testing) allows for several combinations
- Hinge, least-squares, quantile, and expectile-solver already implemented (more to come)
- Architecture aims at re-use of parts of the software for other learning algorithms

# The user view

# Main Components

## Binaries

- Can be found in the folder `bin`
- Main programs are:
  `svm-train`, `svm-select`, and `svm-test`
- Some tools to analyze data and create train/test partitions etc.

# Main Components

## Binaries

- Can be found in the folder `bin`
- Main programs are:
  `svm-train`, `svm-select`, and `svm-test`
- Some tools to analyze data and create train/test partitions etc.

## Scripts

- Can be found in the folder `scripts`
- Are designed for quick or unexperienced usage
- Cover (binary-, multiclass-, Neyman-Pearson-) classification, least squares, quantile, and expectile regression, Mona's Voronoi partitions, bootstrapping

# Data Formats

## Supported Formats

- Comma-separated values `.csv`
- LIBSVMs format `.lsv`
- Some others (similar to `.csv`)

# Data Formats

## Supported Formats

- Comma-separated values `.csv`
- LIBSVMs format `.lsv`
- Some others (similar to `.csv`)

## Comma-separated values

$$
\begin{array}{llll}
y_1, & x_{1,1}, & \ldots, & x_{1,d} \\
\vdots & & & \\
y_n, & x_{n,1}, & \ldots, & x_{n,d}
\end{array}
$$

# Quick Usage with Scripts

## Generalities

- Can be found in the folder `scripts`
- Are called by e.g. `./mc-svm.sh <basename>`
- Perform parameter selection by 5-fold cross-validation on 10x10 grid and test resulting decision functions
- Simple help function by typing e.g. `./mc-svm.sh`

# Quick Usage with Scripts

## Generalities

- Can be found in the folder `scripts`
- Are called by e.g. `./mc-svm.sh <basename>`
- Perform parameter selection by 5-fold cross-validation on 10x10 grid and test resulting decision functions
- Simple help function by typing e.g. `./mc-svm.sh`
- Reads the following files from the folder `data`:
  `<basename>.train.csv`
  `<basename>.test.csv`

# Quick Usage with Scripts

## Generalities

- Can be found in the folder `scripts`
- Are called by e.g. `./mc-svm.sh <basename>`
- Perform parameter selection by 5-fold cross-validation on 10x10 grid and test resulting decision functions
- Simple help function by typing e.g. `./mc-svm.sh`
- Reads the following files from the folder `data`:
  `<basename>.train.csv`
  `<basename>.test.csv`
- Writes the following files into the folder `results`:
  `<basename>.train.log` `<basename>.train.aux`
  `<basename>.select.log` `<basename>.sol`
  `<basename>.test.log` `<basename>.result.csv`

# List of Scripts

- (Multiclass) classification (default AvA): `mc-svm.sh`
- Least squares regression: `ls-svm.sh`
- Quantile regression: `qt-svm.sh`
- Expectile regression: `ex-svm.sh`
- Neyman-Pearson classification: `npl-svm.sh`
- Voronoi partitions: `vp-svm.sh`
- Boot Strapping: `bs-svm.sh`

# Experienced Usage with Binaries

**Generalities**

- Can be found in the folder `bin`
- Simple help function by typing e.g. `./svm-train`
- Detailed help function by typing e.g. `./svm-train -<option-name>`

# Experienced Usage with Binaries

**Generalities**

- Can be found in the folder `bin`
- Simple help function by typing e.g. `./svm-train`
- Detailed help function by typing e.g. `./svm-train -<option-name>`
- Data flow between the different programs

# Experienced Usage with Binaries

**Generalities**

- Can be found in the folder `bin`
- Simple help function by typing e.g. `./svm-train`
- Detailed help function by typing e.g. `./svm-train -<option-name>`
- Data flow between the different programs
- Format of the output files

# Implementational aspects

# Overview

**The code of the library consists of three parts**

- Shared code (documented and cleaned)
  - I/O-operations
  - Hardware support: SIMD, threads, GPUs
  - Data set handling: tasks, cells, folds
  - Cross-validation
  - Decision functions
- SVM-specific code
  - Solvers
  - Main programs
  - Decision functions
- Tools
  - Data set analysis
  - Data set manipulation

# Offset in the Primal

**With or without offset?**

- Our SVM formulation is

$$f_{D,\lambda} = \arg\min_{f \in H} \ \lambda \|f\|_H^2 + \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, f(x_i)\big) \ ,$$

- The standard one is

$$f_{D,\lambda} = \arg\min_{(f,b) \in H \times \mathbb{R}} \ \lambda \|f\|_H^2 + \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, f(x_i) + b\big) \ ,$$

- The offset $b$ usually leads to an equality constraint in the dual optimization problem.

# Offset in the Dual

## Dual optimization problem for the hinge loss

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq \frac{1}{2\lambda n} .$$

## Re-substitution

$$f_{D,\lambda} = \sum_{i=1}^{n} y_i \alpha_i^* k(\,\cdot\,, x_i)\,,$$

# Offset in the Dual

## Dual optimization problem for the hinge loss

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq \frac{1}{2\lambda n} .$$

## Re-substitution

$$f_{D,\lambda} = \sum_{i=1}^{n} y_i \alpha_i^* k(\,\cdot\,, x_i)\,,$$

## SMO (Platt, 1999)

- Select working set $I \subset \{1, \ldots, n\}$.
- Optimize over the variable $\alpha_i$, $i \in I$.
- Repeat until stopping criterion is satisfied.

# Working Set Selection I

**With offset**

- Equality constraint $\sum_{i=1}^{n} y_i \alpha_i = 0$ requires $|I| \geq 2$.
- Not all pairs $(i, j)$ are possible.
- Practice: MVP or second order methods.

# Working Set Selection I

**With offset**

- Equality constraint $\sum_{i=1}^{n} y_i \alpha_i = 0$ requires $|I| \geq 2$.
- Not all pairs $(i, j)$ are possible.
- Practice: MVP or second order methods.

**Without offset**

- No equality constraint $\implies$ $|I| = 1$ is sufficient (but not efficient).
- All pairs $(i, j)$ are possible, but this requires $O(n^2)$ search!

# Working Set Selection II

**Our approach:**

- Search for $i$ with maximal gain in the dual.
- For $j$ consider the following candidates:
  - The $i$ of previous iteration.
  - The index with second best gain in the dual.
  - The indices belonging to the 10-15 nearest neightbors $x_j$ of $x_i$.
- Pick the candidate whose 2D-gain in the dual is maximal.
- This requires one cheap $O(n)$ pass through the indices.

# Further Improvements

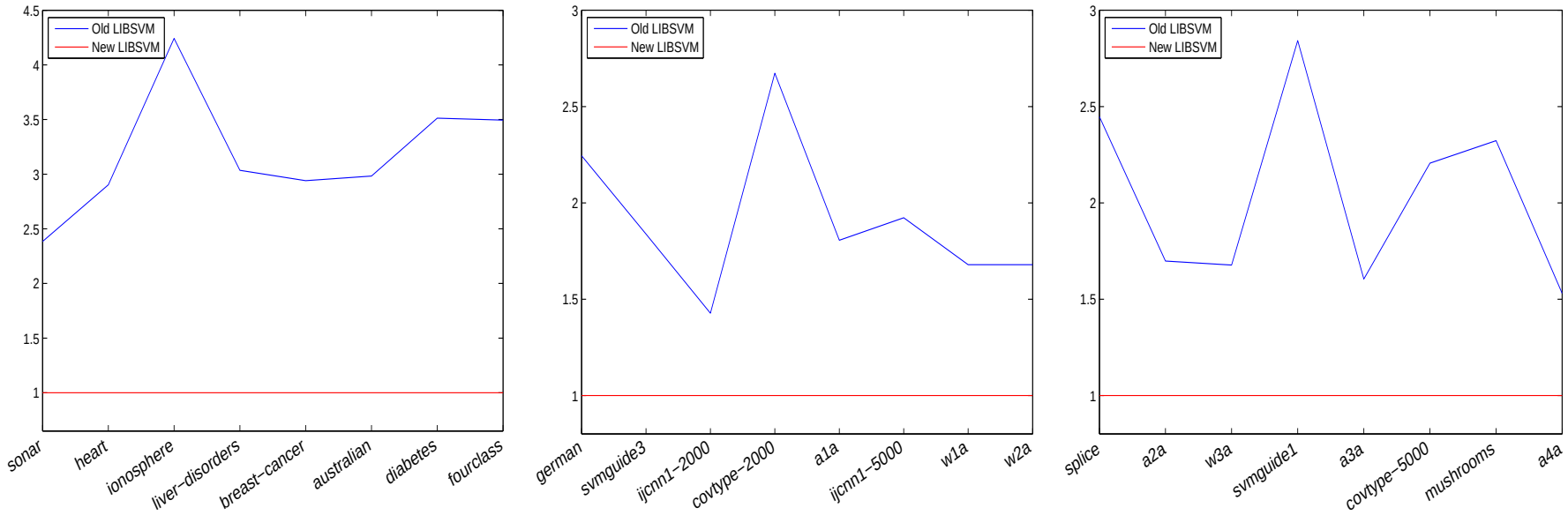
**Some more aspects:**

- Missing equality constraint allows different initializations.
- More warm start options for grid search.
- All inner loops can be vectorized using SIMD instructions.
  $\rightsquigarrow$ Vectorization and no if-branches
- Statistical analysis suggests an aggressive stopping criterion

# Some comparisons

# Runtimes I

## Original LIBSVM vs. SSE2-vectorized code.

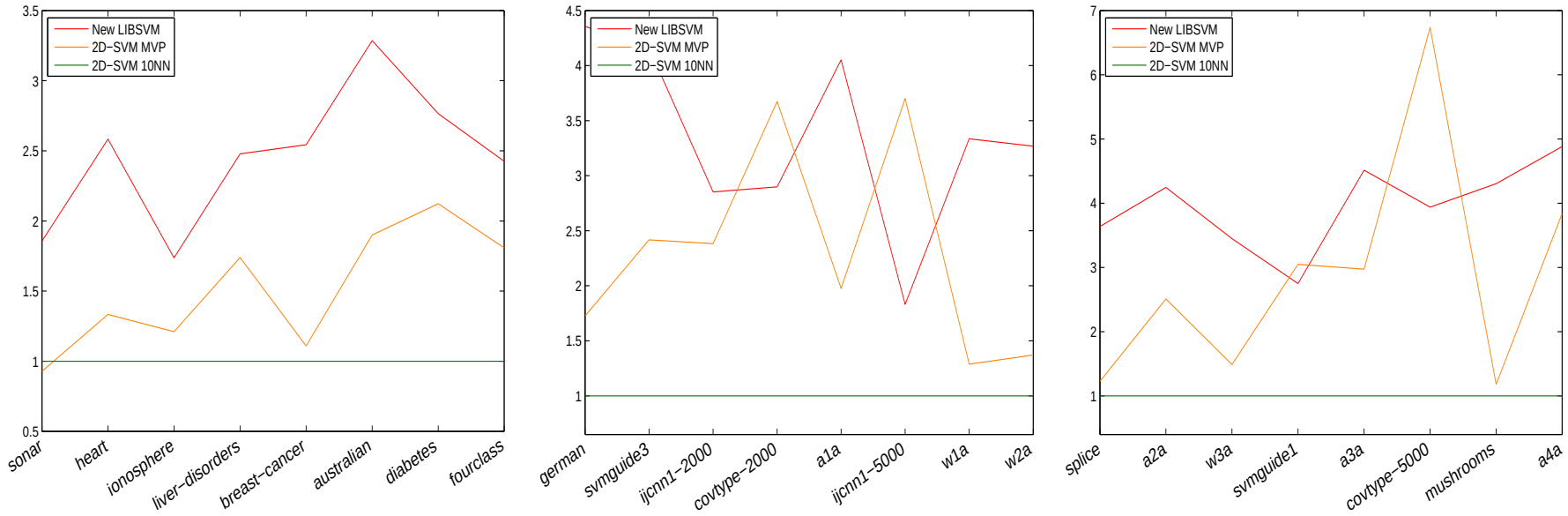



## Conclusion.

Better kernel handling and partial vectorization yields an improvement by a factor of 1.5 - 2.

# Runtimes II

**Improved LIBSVM vs. our solver with cold start**

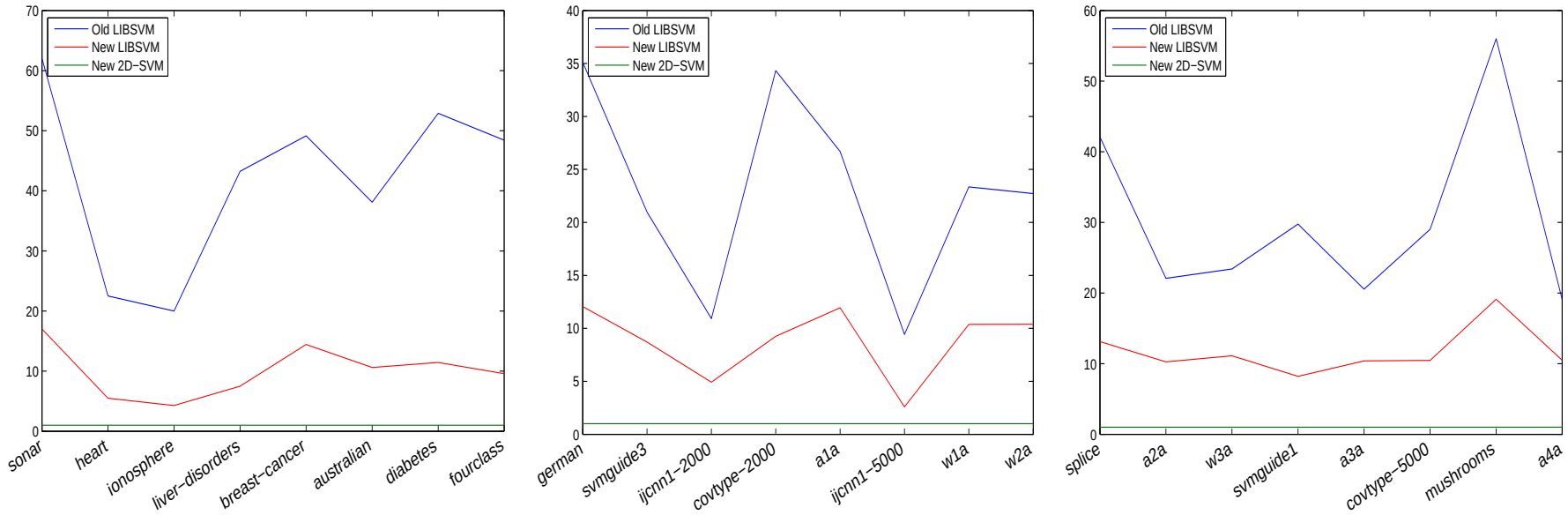



**Conclusion.**

Often an improvement by a factor of 4, partially because less iterations, partially because each iteration is cheaper.

# Improving the Training Time III

**Grid search with warm start.**

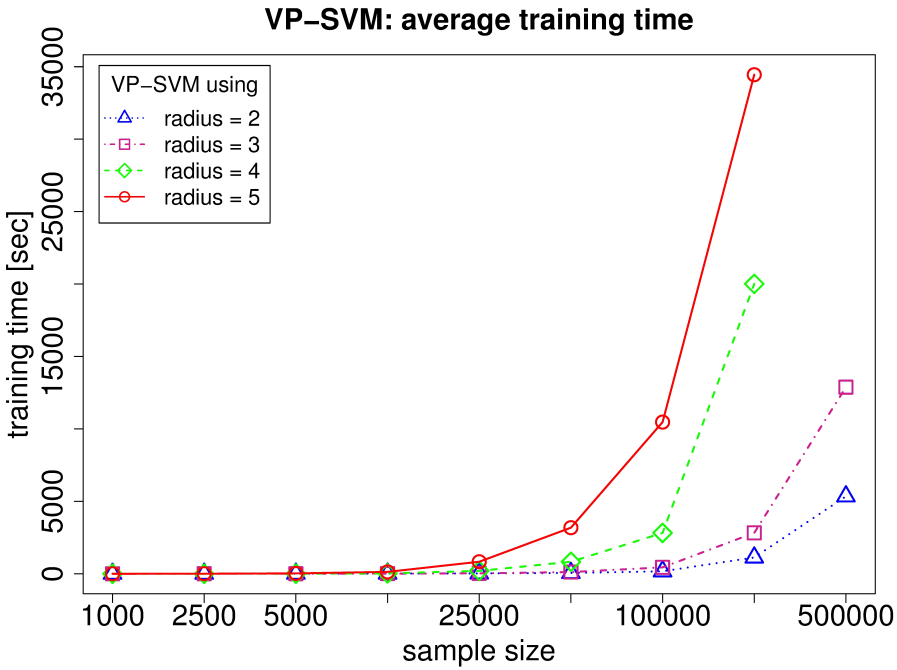


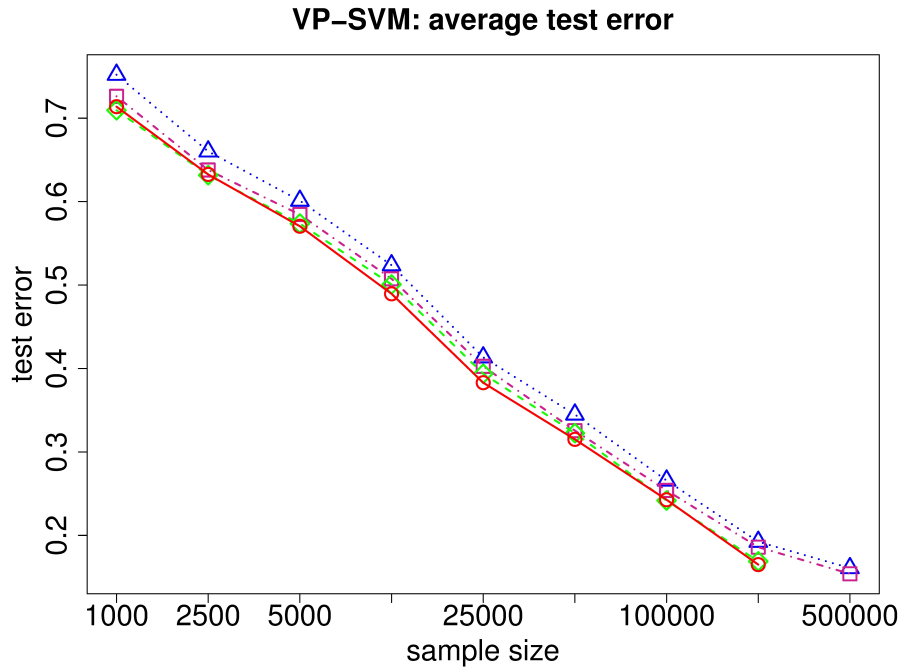

**Conclusion.**

Our new SVM solver is usually at least 20 times faster than off-the-shelf LIBSVM when hyper-parameters are determined by grid-search.

# Different Radii

- **Data:** covertype from LIBSVM site, $d = 54$



# VP vs. Random Chunks

- **Data:** covertype from LIBSVM
- **Method:** Fastest VP-SVM and two similarly fast SVMs on chuncks.

