B: A Comparative Study on Kernelized Support Vector Machines

Daniel Horn
Aydın Demircioğlu
Bernd Bischl
Tobias Glasmachers
Claus Weihs

TU Dortmund / Ruhr-Universität Bochum

Colchester, July 7, 2014
Who we are

Project ‘Large Scale Support Vector Learning’
with staff members from

TU Technische Universität Dortmund

RUHR Universität Bochum

RUB

INI

funded by MERCUR

Webpage at http://www.largescalesvm.de
Support Vector Machines

- Large-margin classifier ...
- ... in a kernel-induced feature space.
- Given data

\((x_1, y_1), \ldots, (x_n, y_n) \in (\mathbb{R}^p \times \{\pm 1\})^n\)

train classifier \(x \mapsto \hat{y} = \text{sign} (f_\alpha(x))\)

with decision function

\[ f_\alpha(x) = \sum_i \alpha_i k(x, x_i) \]

through minimization of regularized empirical risk

\[
\min_{\alpha} \quad \frac{1}{2} \alpha^T K \alpha + C \cdot \sum_{i=1}^{n} \max \{0, 1 - y_i f_\alpha(x_i)\}
\]

where \(K_{ij} = k(x_i, x_j)\) kernel Gram matrix, \(C\) penalty parameter.

- Gaussian kernel

\[ k(x, x') = \exp(-\gamma \|x - x'\|^2), \quad \gamma \text{ kernel-width.} \]
Complexity of Support Vector Machine training

- **Problem:**
  Complexity of SVM training is $O(n^3)$

- **Example:**
  Training of LIBSVM on \{5000, 10000, ..., 250000\} samples of the poker dataset

- **Solution:**
  Approximate SVM training

- **Problem:**
  Which of the many approximation algorithms to use?
## Approximative SVM Training Algorithms

<table>
<thead>
<tr>
<th>SVM solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBSVM</td>
<td>“Exact” SMO solver</td>
</tr>
<tr>
<td>BSGD</td>
<td>Budgeted Stochastic Gradient Descent</td>
</tr>
<tr>
<td>LASVM</td>
<td>Online variant of SMO solver</td>
</tr>
<tr>
<td>LIBBVM/CVM</td>
<td>Minimum Enclosing Ball (only squared hinge loss)</td>
</tr>
<tr>
<td>LLSVM</td>
<td>Low-rank kernel approximation + linear solver</td>
</tr>
<tr>
<td>Pegasos</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>SVMperf</td>
<td>Cutting Plane Algorithm</td>
</tr>
</tbody>
</table>
Our project

- **We expect**: Every solver has a trade-off between training time and prediction error: More time for a solver (should) result in a better solution.

- **Our goal**: Analyze this trade-off! Solve the multi-criteria optimization problem with respect to the two objectives error and training time by varying the parameters.

- **The challenge**: Optimizing 2 expensive objectives in a 4-dimensional parameter space.

- **Our approach**: Replace standard grid search with more sophisticated PAREGO-algorithm.
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $y = f(\mathcal{D})$

while evaluation budget not exceeded do
  fit surrogate on $\mathcal{D}$
  get new design point $x^*$ by optimizing an infill criterion
  evaluate new point $y^* = f(x^*)$, update $\mathcal{D}$ and $y$
end while

return $y_{min} = \min y$ and the associated $x_{min}$
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $\mathbf{y} = f(\mathcal{D})$

while evaluation budget not exceeded do
  fit surrogate on $\mathcal{D}$
  get new design point $\mathbf{x}^*$ by optimizing an infill criterion
  evaluate new point $y^* = f(\mathbf{x}^*)$, update $\mathcal{D}$ and $\mathbf{y}$
end while

return $y_{\text{min}} = \min \mathbf{y}$ and the associated $\mathbf{x}_{\text{min}}$

\[
\begin{align*}
\cos(x) & \quad \text{for } x = 0, 2, 4, 6, 8, 10, 12
\end{align*}
\]
The EGO-Algorithm

generate initial design \( D \subset \mathcal{X} \), compute \( y = f(D) \)

\textbf{while} evaluation budget not exceeded \textbf{do}

\hspace{1em} fit surrogate on \( D \)

\hspace{1em} get new design point \( x^* \) by optimizing an infill criterion

\hspace{1em} evaluate new point \( y^* = f(x^*) \), update \( D \) and \( y \)

\textbf{end while}

\textbf{return} \( y_{min} = \min y \) and the associated \( x_{min} \)
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $\mathbf{y} = f(\mathcal{D})$

**while** evaluation budget not exceeded **do**

  fit surrogate on $\mathcal{D}$

  get new design point $\mathbf{x}^*$ by optimizing an infill criterion

  evaluate new point $\mathbf{y}^* = f(\mathbf{x}^*)$, update $\mathcal{D}$ and $\mathbf{y}$

**end while**

**return** $y_{\text{min}} = \min \mathbf{y}$ and the associated $\mathbf{x}_{\text{min}}$
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $\mathbf{y} = f(\mathcal{D})$

while evaluation budget not exceeded do
  fit surrogate on $\mathcal{D}$
  get new design point $\mathbf{x}^*$ by optimizing an infill criterion
  evaluate new point $\mathbf{y}^* = f(\mathbf{x}^*)$, update $\mathcal{D}$ and $\mathbf{y}$
end while

return $y_{\min} = \min \mathbf{y}$ and the associated $x_{\min}$
The EGO-Algorithm

generate initial design \( D \subset \mathcal{X} \), compute \( y = f(D) \)

\[ \textbf{while evaluation budget not exceeded do} \]

\[ \text{fit surrogate on } D \]

\[ \text{get new design point } x^* \text{ by optimizing an infill criterion} \]

\[ \text{evaluate new point } y^* = f(x^*), \text{ update } D \text{ and } y \]

\[ \textbf{end while} \]

\[ \textbf{return } y_{min} = \min y \text{ and the associated } x_{min} \]
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $y = f(\mathcal{D})$

while evaluation budget not exceeded do

    fit surrogate on $\mathcal{D}$

    get new design point $x^*$ by optimizing an infill criterion

    evaluate new point $y^* = f(x^*)$, update $\mathcal{D}$ and $y$

end while

return $y_{\text{min}} = \min y$ and the associated $x_{\text{min}}$
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $y = f(\mathcal{D})$

while evaluation budget not exceeded do
  fit surrogate on $\mathcal{D}$
  get new design point $x^*$ by optimizing an infill criterion
  evaluate new point $y^* = f(x^*)$, update $\mathcal{D}$ and $y$
end while
return $y_{\text{min}} = \min y$ and the associated $x_{\text{min}}$

\begin{align*}
\cos(x) & \quad 0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \\
-1.0 & \quad 0.0 \quad 1.0
\end{align*}
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $\mathbf{y} = f(\mathcal{D})$

\begin{itemize}
  \item \textbf{while} evaluation budget not exceeded \textbf{do}
    \begin{itemize}
      \item fit surrogate on $\mathcal{D}$
      \item get new design point $\mathbf{x}^*$ by optimizing an infill criterion
      \item evaluate new point $\mathbf{y}^* = f(\mathbf{x}^*)$, update $\mathcal{D}$ and $\mathbf{y}$
    \end{itemize}
  \end{itemize}

\textbf{end while}

\textbf{return} $\mathbf{y}_{min} = \min \mathbf{y}$ and the associated $\mathbf{x}_{min}$
The EGO-Algorithm

generate initial design \( \mathcal{D} \subset \mathcal{X} \), compute \( \mathbf{y} = f(\mathcal{D}) \)

\textbf{while} \textit{evaluation budget not exceeded} \textbf{do}

fit surrogate on \( \mathcal{D} \)

get new design point \( \mathbf{x}^* \) by optimizing an infill criterion

evaluate new point \( \mathbf{y}^* = f(\mathbf{x}^*) \), update \( \mathcal{D} \) and \( \mathbf{y} \)

\textbf{end while}

\textbf{return} \( y_{\text{min}} = \min \mathbf{y} \) and the associated \( \mathbf{x}_{\text{min}} \)
The EGO-Algorithm

generate initial design $\mathcal{D} \subset \mathcal{X}$, compute $y = f(\mathcal{D})$

while evaluation budget not exceeded do
    fit surrogate on $\mathcal{D}$
    get new design point $x^*$ by optimizing an infill criterion
    evaluate new point $y^* = f(x^*)$, update $\mathcal{D}$ and $y$
end while

return $y_{\text{min}} = \min y$ and the associated $x_{\text{min}}$
The PAREGO-Algorithm

- **Expected Improvement**: inherent exploration-exploitation trade-off between best point (according to model) and unvisited regions.

- **Extension for multiple criteria**: $f(x) = y = (y_1, \ldots, y_k)$ and we want to reach the Pareto front.

**Approach**: in each iteration, optimize the scalarized problem

$$y^* = \max_j (\lambda_j \cdot y_j) + \rho \cdot \sum_j \lambda_j \cdot y_j,$$

where the weight vector $\lambda$ is newly sampled in each iteration.

- **Parallel version**: many preferably different weight vectors evaluated in parallel on a super-computer.

- **Our Setting**
  - 10 iterations of PAREGO with 20 proposed $\lambda$-points each
  - 50% training data set, 25% test, 25% validation: test error reported
  - restriction from our super-computer: maximum walltime of 8 hours
  - NAs imputed with high values
The design of our study

The parameters \((C, \gamma)\) of the SVM itself were optimized over \(2^{-15,15}\) respectively. Every solver has further approximation parameters:

<table>
<thead>
<tr>
<th>SVM solver</th>
<th>Parameters</th>
<th>Optimization Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pegasos</td>
<td>#Epochs</td>
<td>(2^{0,7})</td>
</tr>
<tr>
<td>BGSD</td>
<td>Budget size, #Epochs</td>
<td>(2^{4,11}, 2^{0,7})</td>
</tr>
<tr>
<td>LLSVM</td>
<td>Matrix rank</td>
<td>(2^{4,11})</td>
</tr>
<tr>
<td>LIBSVM</td>
<td>(\epsilon) (Accuracy = duality gap)</td>
<td>(2^{-13,-1})</td>
</tr>
<tr>
<td>LASVM</td>
<td>(\epsilon) (Accuracy), #Epochs</td>
<td>(2^{-13,-1}, 2^{0,7})</td>
</tr>
<tr>
<td>LIBBVM/CVM</td>
<td>(\epsilon) (Accuracy)</td>
<td>(2^{-19,-1})</td>
</tr>
<tr>
<td>SVMperf</td>
<td>(\epsilon) (Accuracy), #Cutting planes</td>
<td>(2^{-13,-1}, 2^{4,11})</td>
</tr>
</tbody>
</table>

Additional parameters set to default values.
## Datasets

<table>
<thead>
<tr>
<th>data set</th>
<th># points</th>
<th># features</th>
<th>class ratio</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>wXa</td>
<td>34 780</td>
<td>300</td>
<td>34.45</td>
<td>95.19 %</td>
</tr>
<tr>
<td>aXa</td>
<td>36 974</td>
<td>123</td>
<td>3.17</td>
<td>88.72 %</td>
</tr>
<tr>
<td>protein</td>
<td>42 153</td>
<td>357</td>
<td>1.16</td>
<td>71.46 %</td>
</tr>
<tr>
<td>mnist</td>
<td>70 000</td>
<td>780</td>
<td>1.04</td>
<td>80.76 %</td>
</tr>
<tr>
<td>vehicle</td>
<td>98 528</td>
<td>100</td>
<td>1.00</td>
<td>0 %</td>
</tr>
<tr>
<td>shuttle</td>
<td>101 500</td>
<td>9</td>
<td>3.70</td>
<td>0.23 %</td>
</tr>
<tr>
<td>spektren</td>
<td>175 090</td>
<td>22</td>
<td>1.25</td>
<td>0 %</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>176 691</td>
<td>22</td>
<td>9.41</td>
<td>40.91 %</td>
</tr>
<tr>
<td>arthrosis</td>
<td>262 142</td>
<td>178</td>
<td>1.19</td>
<td>0.01 %</td>
</tr>
<tr>
<td>cod-rna</td>
<td>488 565</td>
<td>8</td>
<td>2.00</td>
<td>0.02 %</td>
</tr>
<tr>
<td>covtype</td>
<td>581 012</td>
<td>54</td>
<td>1.05</td>
<td>78 %</td>
</tr>
<tr>
<td>poker</td>
<td>1 025 010</td>
<td>10</td>
<td>1.00</td>
<td>0 %</td>
</tr>
</tbody>
</table>

**Table:** Overview of the data sets.
Test error landscape (LIBSVM)

- **Error of LIBSVM on ijcnn1**: Shows a scatter plot with cost $C$ on the x-axis and $\gamma$ on the y-axis, indicating the error rate for various parameter settings.

- **Execution Time of LIBSVM on ijcnn1**: Demonstrates the execution time for different parameter combinations, with $C$ and $\gamma$ as axes.

The plots highlight the landscape of errors and execution times, allowing for an analysis of parameter space optimization for SVM models.
LIBSVM Pareto front: impact of accuracy parameter $\varepsilon$

Table: Sizes of LIBSVM Pareto-Fronts and min / max of the corresponding epsilon values.

<table>
<thead>
<tr>
<th>dataset</th>
<th>n</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>wXa</td>
<td>6</td>
<td>0.0030</td>
<td>0.2440</td>
</tr>
<tr>
<td>aXa</td>
<td>4</td>
<td>0.2383</td>
<td>0.4997</td>
</tr>
<tr>
<td>protein</td>
<td>4</td>
<td>0.1337</td>
<td>0.4945</td>
</tr>
<tr>
<td>mnist</td>
<td>4</td>
<td>0.0011</td>
<td>0.0637</td>
</tr>
<tr>
<td>vehicle</td>
<td>2</td>
<td>0.0031</td>
<td>0.0102</td>
</tr>
<tr>
<td>shuttle</td>
<td>4</td>
<td>0.0003</td>
<td>0.1846</td>
</tr>
<tr>
<td>spektren</td>
<td>1</td>
<td>0.4996</td>
<td></td>
</tr>
<tr>
<td>ijcnn1</td>
<td>6</td>
<td>0.0003</td>
<td>0.0213</td>
</tr>
<tr>
<td>arthrosis</td>
<td>4</td>
<td>0.0047</td>
<td>0.0126</td>
</tr>
<tr>
<td>cod-rna</td>
<td>11</td>
<td>0.0001</td>
<td>0.0434</td>
</tr>
<tr>
<td>covtype</td>
<td>7</td>
<td>0.0001</td>
<td>0.4525</td>
</tr>
</tbody>
</table>
All Pareto fronts for ijcnn1 dataset

- 176,691 samples
- 22 features
- 9.41 class ratio
- LIBSVM: exact but slow
- LIBBVM/CVM: good front, speed increase with small accuracy loss
- SVMperf / LLSVM / BSGD: slower
- LASVM / Pegasos: less exact and even slower than LIBSVM
All Pareto fronts for protein dataset

- LIBSVM: exact but slow
- LASVM: more exact, but even slower
- Pegasos: slow
- all others: slow or inaccurate
All Pareto fronts for the poker dataset

- LIBSVM, Pegasos and LASVM expired completely
- LLSVM, LIBBVM/CVM: fast but inaccurate
- SVMperf, BSGD: good front, slow, but low error.

1 025 010 samples
10 features
1.00 class ratio

10000
eexecTime
error

<table>
<thead>
<tr>
<th>solver</th>
<th>BSGD</th>
<th>LLSVM</th>
<th>libBVM</th>
<th>libCVM</th>
<th>SVMperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>0.39</td>
<td>0.42</td>
<td>0.45</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>execTime</td>
<td>1</td>
<td>0.25</td>
<td>0.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 1 025 010 samples
- 10 features
- 1.00 class ratio

- LIBSVM, Pegasos and LASVM expired completely
- LLSVM, LIBBVM/CVM: fast but inaccurate
- SVMperf, BSGD: good front, slow, but low error.
Conclusion

Contribution:
- Analysis of the quality-runtime trade-off of different solvers.

Results:
- No clear winner, depends on the data set.
- BSGD and SVMperf promising on large problems.
- LLSVM really fast.
- LIBSVM gives high quality solutions (expected), tuning of accuracy parameter may make sense.
- LIBCVM/BVM not reliable.
- LASVM and Pegasos not competitive.
Future research

- More solvers,
- more parameters,
- more problems.
- Even better multi-objective parameter tuning.
- Add multi-class and regression problems.
- Experimental setup without super computer.

- Subsampling (see first part)
- Relate empirical results to theory.
- Improve existing approximative solvers and think about new ones.


