Support Vector Machines: Algorithms and Applications

Tong Wen
LBNL
in collaboration with
Alan Edelman
MIT

Jan 17, 2003
Overview

- Support Vector Machines (SVMs) solve classification problems by learning from examples.

- Contents:
  1. Introduction to Support Vector Machines.
  2. Fast SVM training algorithms.
  3. Financial applications of SVMs?

- Notations:

\[ x_i \] the \( i \)th training point
\[ y_i \] the label of the \( i \)th training point
\[ \alpha \] the vector of Lagrange multipliers
\[ \alpha_i \] the \( i \)th Lagrange multiplier
\[ w \] the vector giving the normal direction of a hyperplane
\[ b \] the bias term
\[ n \] the dimension of a training point
\[ m \] the number of training points
\[ H \] the Hessian matrix
\[ X \] the matrix whose columns are the training points
\[ P \] the matrix that spans the current search space
\[ Q \] the orthogonal complement of \( P \)
\[ S_1 \] the set of SVs that are correctly separated
\[ S_2 \] the set of SVs that are separated with errors
\[ h(\cdot) \] a decision rule (function)
Introduction to Support Vector Machines

- An example.

- The primal SVM quadratic programming problem.

- The dual SVM quadratic programming problem.

- Properties of Support Vectors(SVs).
An Example

- The mechanism that generates each point \( x \) and its color \( y \):

- The main theme of classification is to determine a decision rule:

\[
h : \mathbb{X} \subseteq \mathbb{R}^n \rightarrow \mathbb{Y} \subseteq \mathbb{R}.
\]

For this example, \( \mathbb{Y} = \{1, -1\} \).

- What is the optimal solution to our detection problem?
An Example

- By Bayes’ Rule, the maximum likelihood decision rule is

\[ h_{ML}(x) = \arg\max_{\hat{y} \in \{\pm1\}} p(x \mid y = \hat{y}), \]

which minimize the risk

\[ R[h] = E\left[\frac{1}{2} | y - h(x) | \right]. \]

\[ m_-(1,1), m_+(3,3), \sigma=1 \]
Support Vector Machines

- The decision rule $h_{ML}(\cdot)$ depends on $P(x, y)$.

- In general, statistical information such as $P(x, y)$ is not available. A distribution-free approach is needed.

- The SVM decision rule is determined based on the training examples:
  \[ S = \{(x_1, y_1), \ldots, (x_m, y_m)\}. \]

- The training error:
  \[ R_S[h] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |y_i - h(x_i)|. \]

- Overfitting vs. the training error.
• In the previous example,

\[ h_{ML}(x) = \text{sgn}(w^T x + b). \]

• Every decision rule geometrically corresponds to a separating boundary.

• If \( f_{SB}(x) = 0 \) defines the separating boundary, then the corresponding decision rule has a general form

\[ h(x) = \text{sgn}(f_{SB}(x)). \]

• The SVM decision rule:

\[
\begin{align*}
 h_{SVM}(x) &= \text{sgn}(w^T x + b), \\
 h_{SVM}(x) &= \text{sgn}(w^T \phi(x) + b) \\
 &= \text{sgn}(\sum_{i=1}^{m} \alpha_i k(x_i, x) + b).
\end{align*}
\]
Support Vector Machines

- SVMs use two parallel hyperplanes instead of one to determine the separating boundary. These two (oriented) hyperplanes are computed directly using the training set $S$.

\[ d = \frac{2}{\|w\|} \]
The Primal SVM QP Problem: 
the Separable Case

- $w$ and $b$ are determined by the quadratic programming (QP) problem:

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

subject to

$$y_i(w^T x_i + b) \geq 1, \text{ for } i = 1, \ldots, m.$$  

- We can always separate the training points by mapping them to a higher dimensional space $\mathbb{R}^{n'}$ ($n' > n$), such that in $\mathbb{R}^{n'}$ the two sets $\{\phi(x_i) \mid y_i = 1\}$ and $\{\phi(x_i) \mid y_i = -1\}$ can be separated by hyperplanes.

- In practice, people may not want to fully separate the training set because of overfitting.
The Primal SVM QP Problem:
the Inseparable Case

- To accommodate the separation errors on $S$, nonnegative slack variables (errors) $\varepsilon_i$ are introduced:

$$\minimize_{w,b,\varepsilon} \quad \frac{1}{2} \|w\|^2 + c \sum \varepsilon_i$$

subject to (for $i = 1, \ldots, m$)

$$y_i(w^T \phi(x_i) + b) \geq 1 - \varepsilon_i \text{ and } \varepsilon_i \geq 0.$$

- If the training point $\phi(x_i)$ is correctly separated by the soft-margin hyperplanes, then $\varepsilon_i = 0$.

- The coefficient $c$ controls the trade-off between the size of the gap and how well $S$ is separated.
The Dual SVM QP Problems

- In practice, the dual problems are computed instead:

\[
\max_{\alpha} \alpha^T 1 - \frac{1}{2} \alpha^T H \alpha
\]

subject to

\[
y^T \alpha = 0, \\
\alpha \geq 0.
\]

- The Hessian matrix \( H_{ij} = y_i (\phi(x_i)^T \phi(x_j)) y_j \) is a \( m \times m \) symmetric positive semi-definite matrix.
The Positive Definite Kernel Functions

- Since only inner products are involved in the dual problems, a positive definite kernel function $k(\cdot, \cdot)$ is used instead of $\phi(\cdot)$, where

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \text{ and } H_{ij} = y_i k(x_i, x_j) y_j.$$  

- Using kernel functions avoids the curse of dimensionality.

- Three positive definite kernels are used here:

  linear: $k(s, t) = s^T t$
  poly($d$): $k(s, t) = (s^T t + 1)^d$
  rbf($\sigma$): $k(s, t) = e^{-\|s-t\|^2/(2\sigma^2)}$

- For simplicity, we use $x_i$ in place of $\phi(x_i)$ in our later discussions.
Support Vectors

- At optimality,

\[ w = \sum_{i=1}^{m} y_i \alpha_i x_i, \]

and if \( 0 < \alpha_i < c \) \( (y_i (w^T x_i + b) = 1) \) then

\[ b = y_i - w^T x_i = y_i (1 - e_i^T H \alpha). \]

- \( h_{\text{SVM}}(x) = \text{sgn}(w^T x + b). \)

- \( S_1 \cup S_2 \) contain the support vectors (SVs).

\[ \begin{cases} 0 < \alpha_i < c & \leftrightarrow x_i \in S_1, \ S_1 = \{ x_i | y_i (w^T x_i + b) = 1 (\varepsilon_i = 0) \}; \\ \alpha_i = c & \leftrightarrow x_i \in S_2, \ S_2 = \{ x_i | y_i (w^T x_i + b) = 1 - \varepsilon_i \ \text{and} \ \varepsilon_i > 0 \}; \\ \alpha_i = 0 & \leftrightarrow x_i \in S_3, \ S_3 = \{ x_i | y_i (w^T x_i + b) > 1 (\varepsilon_i = 0) \}. \]

- SVs are the training points that determine the two separating hyperplanes through linear equations.

- The solution to the dual problem is sparse.
Solving the Dual SVM QP Problem Efficiently

- Solving the dual problem is computationally challenging.
- The decomposition strategy.
- A projected conjugate gradient algorithm.
- Speed considerations.
- Numerical experiments.
- Conclusion and future work.
The Decomposition Strategy

- The sparsity property.

- An observation from the primal problem point of view:

- How to apply this strategy to solve the dual problem efficiently?
  1. Constructing a subproblem.
  2. Solving the subproblems.
  3. The optimality conditions.

- A projected conjugate gradient algorithm.
Applying the Decomposition Strategy

- Projecting the dual QP problem onto a subspace.
  - Let us restrict $\alpha$ in the affine space determined by $\alpha_0$ and $P$. That is, $\alpha = \alpha_0 + P \hat{\alpha}$.
  - The projected dual problem is

$$\max_{\hat{\alpha}} F(\hat{\alpha}) \equiv \hat{\alpha}^T P^T r_0 - \frac{1}{2} \hat{\alpha}^T P^T H P \hat{\alpha}$$

subject to

$$y^T P \hat{\alpha} = 0,$$
$$0 \leq \alpha_0 + P \hat{\alpha} \leq c,$$

where $r_0$ is the gradient of $F(\alpha)$ at $\alpha_0$.

- $P$ is a general matrix so far. It can be specified directly or by its orthogonal complement $Q$, where $P^T Q = 0$.

- How to determine $P$ or $Q$, and how to solve each subproblem?
Constructing a Subproblem

- $\alpha = \alpha_0 + P\hat{\alpha}$.

<table>
<thead>
<tr>
<th>$P = E_p$ and $Q = E_q$ ($m = p + q$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize $\hat{F}(\hat{\alpha}) \equiv \hat{\alpha}^T \hat{r}_0 - \frac{1}{2} \hat{\alpha}^T \hat{H} \hat{\alpha}$</td>
</tr>
<tr>
<td>subject to $\hat{y}^T \hat{\alpha} = 0,$</td>
</tr>
<tr>
<td>$0 \leq \alpha_0 + \hat{\alpha} \leq c,$</td>
</tr>
<tr>
<td>where $\hat{r}_0 = P^T r_0, \hat{y} = P^T y, \hat{H} = P^T H P$, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P = I - \frac{\hat{y}\hat{y}^T}{p}, Q = \hat{y}$ and $\hat{\alpha} = P\hat{\alpha}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize $\hat{F}(\hat{\alpha}) \equiv \hat{\alpha}^T P^T \hat{r}_0 - \frac{1}{2} \hat{\alpha}^T P^T \hat{H} P \hat{\alpha}$</td>
</tr>
<tr>
<td>subject to $0 \leq \alpha_0 + P\hat{\alpha} \leq c.$</td>
</tr>
</tbody>
</table>

- $\hat{H}$ is a $p \times p$ matrix ($p \ll m$).
Solving the Subproblems by the CG Method

- If $0 \leq \hat{\alpha}_0 + P\hat{\alpha} \leq c$ never becomes active, we have
  \[ P^T \hat{H} P \hat{\alpha} = P^T \hat{r}_0, \]
or equivalently
  \[ P \hat{H} P \hat{\alpha} = P \hat{r}_0 \text{ (} P \text{ is symmetric).} \]

<table>
<thead>
<tr>
<th>The CG iteration</th>
<th>The simplified version</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\alpha}^{(0)} = 0$, $\gamma^{(0)} = P\hat{r}_0$,</td>
<td>$\hat{\alpha}^{(0)} = 0$, $\gamma^{(0)} = P\hat{r}_0$,</td>
</tr>
<tr>
<td>$\rho^{(0)} = \gamma^{(0)}$</td>
<td>$\rho^{(0)} = \gamma^{(0)}$</td>
</tr>
<tr>
<td>$\lambda = \frac{\gamma^{(0)} - \rho^{(0)}}{\rho^{(l-1)} - \hat{H} \rho^{(l-1)}}$</td>
<td>$\lambda = \frac{\gamma^{(l-1)} - \rho^{(l-1)}}{\rho^{(l-1)} - \hat{H} \rho^{(l-1)}}$</td>
</tr>
<tr>
<td>$\hat{\alpha}^{(l)} = \hat{\alpha}^{(l-1)} + \lambda \rho^{(l-1)}$</td>
<td>$\hat{\alpha}^{(l)} = \hat{\alpha}^{(l-1)} + \lambda \rho^{(l-1)}$</td>
</tr>
<tr>
<td>$\gamma^{(l)} = \gamma^{(l-1)} - \lambda P \hat{H} P \rho^{(l-1)}$</td>
<td>$\gamma^{(l)} = \gamma^{(l-1)} - \lambda P \hat{H} \rho^{(l-1)}$</td>
</tr>
<tr>
<td>$\mu = \frac{\gamma^{(l)} - \gamma^{(0)}}{\rho^{(l-1)} - \gamma^{(0)}}$</td>
<td>$\mu = \frac{\gamma^{(l)} - \gamma^{(0)}}{\rho^{(l-1)} - \gamma^{(0)}}$</td>
</tr>
<tr>
<td>$\rho^{(l)} = \gamma^{(l)} + \mu \rho^{(l-1)}$</td>
<td>$\rho^{(l)} = \gamma^{(l)} + \mu \rho^{(l-1)}$</td>
</tr>
</tbody>
</table>

- How about $0 \leq (\hat{\alpha}_0)_i + (P\hat{\alpha})_i \leq c$ becomes active during the iteration?
The Optimality Conditions

- An equivalent problem:
  \[ Q = [E_q, y] \text{ and } P = I - Q(Q^T Q)^{-1} Q^T. \]

- Define
  \[ d = Pr \]
  and
  \[ \beta = -[I_{q \times q}, 0](Q^T Q)^{-1} Q^T r, \]
  where \( r = 1 - H\alpha \) and \( P \) is a \( m \times m \) matrix.

- \( \alpha \) is optimal iff
  \[ d = 0 \text{ and } \beta \geq 0. \]

- Since \( (Q^T Q)^{-1} = \begin{bmatrix} I + \frac{E_q y^T E_q}{p} & -
\frac{E_q^T y}{1} \\
-
\frac{y^T E_q}{p} & \frac{p}{1} \end{bmatrix} \), it is easy to compute \( d \) and \( \beta \).

  \[ \beta = E_q^T r + \frac{E_q^T y((E_q^T y)E_q^T r - y^T r)}{p}. \]

- Each time when active constraints are to be relaxed, we start with the one that has the most negative \( \beta_i \).
A Projected Conjugate Gradient Algorithm

- A summary of our algorithm:
  \[ \begin{align*}
  &\alpha = 0; \\
  &\text{initialize } E_p; \\
  \textbf{while } d \neq 0 \text{ or } \beta < 0 \\
  &\text{at most } k \text{ steps of the CG iterations for the subproblem} \\
  &\text{update } \alpha \text{ and } r: \\
  &\quad \alpha = \alpha + E_p(I - \frac{yy^T}{p})\tilde{\alpha}; \\
  &\quad r = r - HE_p(I - \frac{yy^T}{p})\tilde{\alpha}; \\
  &\text{relax at most } l \text{ active constraints with the most negative } \beta_i; \\
  &\text{update } E_p; \\
  &\text{compute the columns of } H \text{ corresponding to the relaxed constraints;} \\
  &\text{update } HE_p \text{ and } \tilde{H}; \\
  \end{align*} \]

- Memory requirement: \((H \text{ is not precomputed!})\)
  \[ X = [x_1, \ldots, x_m] \text{ and } HE_p. \]

- Flops:
  Solving a sequence of smaller problems is cheaper when the solution is sparse.

- Two parameters: \(k\) and \(l\).

- How to initialize \(E_p\)?
Speed Considerations:
Being Adaptive to the Memory Hierarchy

- A model of the modern computer memory hierarchy:

<table>
<thead>
<tr>
<th>level</th>
<th>clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>1</td>
</tr>
<tr>
<td>L1 cache</td>
<td>2 – 3</td>
</tr>
<tr>
<td>L2 cache</td>
<td>6 – 12</td>
</tr>
<tr>
<td>near main memory</td>
<td>6 – 100</td>
</tr>
<tr>
<td>far main memory</td>
<td>100 – 250</td>
</tr>
<tr>
<td>distributed main memory</td>
<td>$O(100)$</td>
</tr>
<tr>
<td>message-passing</td>
<td>$O(1000) – O(10000)$</td>
</tr>
</tbody>
</table>

- Linear algebra operations: level 1 – 3.

- ATLAS BLAS.

- Thanks to MATLAB’s inclusion of ATLAS BLAS.
Speed Considerations: Setting $k$ and $l$ Adaptively

- The spectrum of $H$.
  - The size of $S_1 \cup S_2$.
  - The convergence of the CG method.

- $k$ is used to control the steps of the CG iterations.
  - There is no need to compute every subproblem exactly.
  - The rank deficiency of $H$.

- $l$ is used to control the number of active constraints relaxed at each time.
  - When $l$ active constraints are relaxed, $[x_{i_1}, \ldots, x_{i_l}]^T X$ is computed to generate the corresponding columns of $H$.
  - If $l$ is set large, then there is more chance of some previously relaxed constraints becoming active again.

- $k$ and $l$ is set to be $\hat{k}$, the estimated number of the leading eigenvalues of $H$.
  - In our algorithm, $\hat{k}$ is set to be the number of iterations for the CG method to reduce the residual norm to a scale of $0.01$ for the initial subproblem.
Numerical Experiments

- The two training sets:

<table>
<thead>
<tr>
<th>Training sets</th>
<th>Digit17</th>
<th>Face</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of the training set</td>
<td>13007</td>
<td>31022</td>
</tr>
<tr>
<td>The positive training points</td>
<td>6742</td>
<td>2901</td>
</tr>
<tr>
<td>The negative training points</td>
<td>6265</td>
<td>28121</td>
</tr>
<tr>
<td>The dimension of the training points</td>
<td>784</td>
<td>361</td>
</tr>
<tr>
<td>The format of the training points</td>
<td>sparse</td>
<td>dense</td>
</tr>
<tr>
<td>Separability</td>
<td>easier</td>
<td>harder</td>
</tr>
</tbody>
</table>

- Comparing against SVM\textsuperscript{light} (C) and SvmFu (C++).

- The choices of $k(s, t)$ and $c$:

<table>
<thead>
<tr>
<th>Training sets: Digit17-6000 and Digit17</th>
</tr>
</thead>
<tbody>
<tr>
<td>l(1)</td>
</tr>
<tr>
<td>r10(10)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Training sets: Face-6000, Face-13901 and Face</th>
</tr>
</thead>
<tbody>
<tr>
<td>l(4)</td>
</tr>
<tr>
<td>p2(0.001)</td>
</tr>
</tbody>
</table>
Numerical Experiments

Using Default Settings  Using Estimated Optimal Settings

Digit17–6000

CPU seconds

Digit17–6000

CPU seconds

Face–6000

CPU seconds

Face–6000

CPU seconds
Numerical Experiments

Comparison of FMSvm with $\text{SVM}^{\text{light}}$ and SvmFu with the estimated optimal parameter settings.
Conclusion and Future Work

• Conclusion:
  1. efficient;
  2. easy to use;
  3. highly portable.

• Future work:
  1. Try FMSvm on more training problems and platforms.
  2. Improve FMSvm's performance and understand better its convergence behavior.
Applying SVMs to Financial Markets

- Identifying problems:
  - A mathematical model is available, but only an approximation.
  - Decisions are made based on experience.

- Constructing training sets:
  - Domain knowledges.
  - Data are from different categories.
  - Scaling the training data.

- Using SVMs to predict the movement of Dow Jones Industrials Average Index (DJIA):
  - The geometric Brownian motion solution.
  - The SVM solution.
  - The comparison between the two solutions.

- Conclusion and future work.
The Geometric Brownian Motion Model

- The weak form of the market-efficiency hypothesis.

- The equations:

\[ dS = \mu S\, dt + \sigma S\, dz. \]

By Itô’s lemma, we have

\[ d\ln S = (\mu - \frac{\sigma^2}{2})\, dt + \sigma\, dz. \]

That is,

\[ \ln \frac{S_T}{S_0} \sim N((\mu - \frac{\sigma^2}{2})T, \sigma^2 T). \]

- We want to predict on Thursday (after the market is closed) whether or not the closing index of DJIA on Friday is higher than its opening index on Monday.
The Dow Jones Industrial Average Index

The Dow Jones Index (02-Dec-1985 to 28-Dec-2001)

The Dow Jones Index Plotted in the Logarithmic Scale
The Geometric Brownian Motion Solution

- The geometric Brownian motion model:
  \[
  \frac{dS}{S} = \mu dt + \sigma dz.
  \]

- Its discrete-time version:
  \[
  \frac{\Delta S}{S} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t}.
  \]
  \[
  \frac{\Delta S}{S} \sim N(\mu \Delta t, \sigma^2 \Delta t).
  \]

- Stock price on Friday: \((R = \frac{S_{Fr} - S_{Th}}{S_{Th}} \sim N(\mu, \sigma^2))\)
  \[
  S_{Fr} = (1 + R)S_{Th}
  = (1 + \mu + \sigma \epsilon)S_{Th}.
  \]

- We want to know if \(S_{Fr} \geq S_{Mo}\).
  \[
  \epsilon \geq \frac{S_{Mo}/S_{Th} - 1 - \mu}{\sigma} = \xi.
  \]

- Since
  \[
  \begin{cases} 
  \Pr(\epsilon > \xi) > \Pr(\epsilon < \xi) & \text{if } \xi < 0 \\
  \Pr(\epsilon > \xi) < \Pr(\epsilon < \xi) & \text{if } \xi > 0,
  \end{cases}
  \]
  \[
  h_{\text{GBM}}(\xi) = \text{sgn}(-\xi).
  \]
The Geometric Brownian Motion Solution

- Estimating $\mu$ and $\sigma$: only $\mu$ matters.

![Graph of Estimated Daily Returns on Fridays (07-Dec-1990 to 28-Dec-2001)](image)


- The performance: 96 errors out of 511 trials. The accuracy of $h_{\text{GBM}}$ on this test set is 81.21%.
The SVM Solution

- \( h_{\text{SVM}}(x) = \text{sgn}(w^T x + b) \), where “experience” is incorporated in \( w \) and \( b \).

  \[ x = [341.3\ 343.6\ 339.4\ 340.9\ 23300\ \ldots\ 340.4\ 343.0\ 339.2\ 341.1\ 30000]^T \]

- Scaling the raw training vectors:
  \[ x_s = [1\ 1.007\ 0.994\ 0.999\ 1\ \ldots\ 0.997\ 1.005\ 0.994\ 0.999\ 1.288]^T \]

- Results: (There are totally 1704 training vectors, and the test set contains 511 vectors.)

<table>
<thead>
<tr>
<th></th>
<th>poly(4)</th>
<th>rbf(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( c )</td>
<td>Errors</td>
</tr>
<tr>
<td>poly(4)</td>
<td>0.2</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>1.6</td>
<td>97</td>
</tr>
<tr>
<td>rbf(1)</td>
<td>20</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>98</td>
</tr>
</tbody>
</table>
The SVM Solution

- Our intuition is that if there is any available information correlated to the movement of DJIA, then adding it to the training set may improve the performance.

- We add the daily federal funds rate to the training set. We choose this rate simply because it is a daily rate.

- Again, we use the relative values with respect to the Monday rate in the training vectors, instead of real rates.

- Results:

<table>
<thead>
<tr>
<th>$c$</th>
<th>poly(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>103</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
</tr>
<tr>
<td>1.5</td>
<td>96</td>
</tr>
<tr>
<td>2.5</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>94</td>
</tr>
<tr>
<td>20</td>
<td>93</td>
</tr>
<tr>
<td>30</td>
<td>89</td>
</tr>
<tr>
<td>35</td>
<td>89</td>
</tr>
</tbody>
</table>

- When $c = 30$, it takes around 2.6 hours (on newton.mit.edu) to train the SVM comparing with the scale of seconds when using the old training set.
The Geometric Brownian Motion Model VS. SVMs

- The Geometric Brownian Motion model is only an approximation.

- The semi-strong form of the market-efficiency hypothesis.

![Graph showing h_{GBM} vs. h_{SVM} from 1991 to 2001](image.png)
Conclusions and Future Work

- **Conclusions:**
  1. SVMs can learn well from time series data.
  2. There are potentials to achieve better performance using SVMs.

- **Future work:**
  1. We want to find good applications of SVMs (need domain knowledge).
  2. How to construct a training set based on data from different categories is also an interesting problem.