Sparse solutions for linear prediction problems

Thursday 27 April 2006

Thesis Defense of Tyler Neylon
Courant Institute

Advisors Dennis Shasha and Mehryar Mohri
Acknowledgements

Advisors

Dennis Shasha
Mehryar Mohri

Also

Richard Cole
Clifford Hurvich
Lee-Ad Gottlieb
Thesis Work

- **Learning theory**
  - VC dimension bounds
  - Sample compression schemes

- **Time series analysis**
  - Matrix unsparsifiability & complexity
  - Incremental algorithms & experiments
    - Exact solutions
    - Approximate solutions
Outline

• Motivation

• Learning Theory
  ✦ Idea of the VC dimension
  ✦ VC dim bounds for sparse classifiers

• Algorithms
  ✦ Complexity
  ✦ Incremental algorithms
Motivation

- Example Data: stock prices
- We might have an equality similar to:
  \[ 2 \text{IBM} = \text{HPQ} + 2 \text{AAPL} \]
- If \( A \) is our data matrix, look for sparse \( x \):
  \[ Ax \approx 0 \]
Motivation

- If $A$ is our data matrix, look for sparse $x$:

$$Ax \approx 0$$

- Advantages:

  - **Select** a small subset of the time series that are interdependent
  - **Predict** a particular time series, using time-shifted data

$$IBM_{t+1} = \frac{1}{2} HPQ_t + AAPL_t$$

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>HPQ</th>
<th>IBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>9am</td>
<td>70</td>
<td>30</td>
<td>83</td>
</tr>
<tr>
<td>68</td>
<td>30</td>
<td></td>
<td>84</td>
</tr>
<tr>
<td>68</td>
<td>32</td>
<td></td>
<td>86</td>
</tr>
<tr>
<td>3pm</td>
<td>71</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

time-shifted data
Motivation: why sparsity?

• In this talk:
  ✦ Subset selection
  ✦ Better learning
  ✦ Lower cost

• In other applications:
  ✦ Compression

Occam’s Razor: among otherwise equal explanations, the simplest is best
Outline

- Motivation
- **Learning Theory**
  - Idea of the VC dimension
  - VC dim bounds for sparse classifiers
- Algorithms
  - Complexity
  - Incremental algorithms
Learning Theory

- Example: diagnosing diabetes
  - body mass index
  - plasma glucose level
  - Given labeled points, can we learn the pattern behind them?

- Example: learning a circle in the plane
  - The + points form the interior of a circle. Which circle?
Learning Theory

Example: diagnosing diabetes

- body mass index
- plasma glucose level

Given labeled points, can we learn the pattern behind them?

Example: learning a circle in the plane

concept or hypothesis
Our raw (unlabeled) data is from a **ground set** $X$

Labeled data are pairs $(x,y)$, where $y \in \{+, -\}$

A **concept** $c$ is a subset of $X$: $c \subset X$

- We think of $c$ as the set of all points labeled $+$

A **concept class** $C$ is a set of concepts
Learning Theory

• Labeled data are pairs \((x, y)\) in \(X \times \{+, -\}\)

• **Assumptions:**
  - Data arrives from a fixed probability distribution on \(X \times \{+, -\}\)
  - Some concept in concept class \(C\) gives a good approximation to this distribution

• **Goal:** based on a small amount of labeled data, choose \(c \in C\) which is likely to give accurate labels \(\{+, -\}\) to future points \(x \in X\)
VC Dimension

named for V. Vapnik and A. Chervonenkis

- General problem in learning: overfitting
- Example: fitting data by a polynomial

- Intuitive goal: Avoid overfitting by reducing the degrees of freedom in choosing our concept
VC Dimension

• *Definition* Given concept class $C$ and subset $Y \subset X$, the **restriction** $C|Y$ is given by

$$C|Y = \{c \cap Y : c \in C\}$$

• *Definition* A subset $Y \subset X$ is **shattered** by concept class $C$ when $C|Y = 2^Y$

**Example**

$C = \{h_1, h_2, h_3, h_4\}$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>${0, 0, 1, 0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_2$</td>
<td>${0, 1, 0, 1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td>${0, 1, 1, 1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_4$</td>
<td>${1, 0, 0, 0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Y = \{c, d\}$ is shattered

$Y = \{a, b\}$ is not shattered
VC Dimension

• **Definition**  Given concept class \( C \subset 2^X \),

\[
VCdim(C) = \max_{Y \subset X} \{|Y| : C \text{ shatters } Y \}
\]

**Example:** \( X = \mathbb{R}^2 \), \( C \) = set of halfplanes

\( VCdim(C) \geq 3 \), \( Y = \) is shattered

\[
\begin{array}{cccc}
- & - & + & - \\
+ & - & - & - \\
+ & + & + & + \\
- & + & - & + \\
- & - & - & - \\
\end{array}
\]

\( VCdim(C) < 4 \)

..but no set of 4 points can be shattered by \( C \)
VC Dimension

• **Definition**  Given concept class \( C \subset 2^X \),

\[
VCdim(C) = \max_{Y \subset X} \{|Y| : C \text{ shatters } Y \}
\]

Example: \( X = \mathbb{R}^2 \) \( C = \text{ set of halfplanes} \) \( \text{..so } VCdim(C)=3 \)

\( VCdim(C) \geq 3, \ Y = \) is shattered

\[
\begin{align*}
\text{Case 1: convex hull = triangle} & \\
\text{Case 2: convex hull = quadrilateral}
\end{align*}
\]
VC Dimension

How does this definition achieve its goal: avoid overfitting the data?

(we’ll see in a moment)
Sparse Linear Classifiers

- **Linear sparsity**: need to define our concept class

\[ X = \mathbb{R}^n \]  
Given \( u \in \mathbb{R}^n \), define concept \( c_u \) by  
\[ c_u = \{ x : x \cdot u \geq 0 \}. \]  
Let \( \| u \|_0 = \# \) nonzeros in \( u \).

Define concept class \( C_k = \{ c_u : \| u \|_0 \leq k \} \).
**Bound on VC Dimension**

**Question:** how does sparsity help to achieve better learning?

- $C_k =$ concept class of sparse linear classifiers

**Theorem**  
If $3 \leq k \leq \frac{9}{20} \sqrt{n}$, then  

$$VCdim(C_k) < 2k \log(n)$$

*Sparser classifiers give better VC dim bounds*
Bound on VC Dimension

• Proof outline: we find a function $f(m)$ so that

\[
\forall Y \subset X, \quad size(C_k|Y) \leq f(|Y|).
\]

Thus

\[
d = VCdim(C_k) \implies \exists Y \text{ of size } d \text{ so that } C|Y = 2^Y
\]

\[
\implies size(C_k|Y) = 2^d \leq f(d)
\]

\[
\implies d \leq \lg (f(d))
\]

Use this inequality to find an upper bound for $d$. 
Bound on VC Dimension

- **Proof outline** - Use Sauer’s lemma to help show that the following function will suffice:

\[
    f(m) = \binom{n}{k} \left( \binom{m}{k} \right) \leq \left( \frac{en}{k} \right)^k \left( \frac{em}{k} \right)^k
\]

where \( \binom{m}{\leq k} = \sum_{i \leq k} \binom{m}{i} \)

Then

\[
    d \leq \lg(f(d))
\]

\[
    \implies d \leq k \lg \left( \frac{e^2 nd}{k^2} \right)
\]

\[
    < k \lg(nd), \quad \text{when } k > e.
\]

Thus

\[
    d - k \lg(d) < k \lg(n).
\]
Bound on VC Dimension

- **Proof outline** - it now suffices to show that

\[ \left( d - k \lg(d) < k \lg(n) \text{ and } 3 \leq k \leq \frac{9}{20} \sqrt{n} \right) \implies d < 2k \lg(n) \]

from last slide

assumed sparsity

our goal

**Parametrize:**

\[ b = \lg(n) \quad z = d - kb \]

Then

\[ z < k \lg(d) = k \lg(z + kb) \implies k > \frac{z}{\lg(z + kb)} \]

so that

\[ z \geq kb \implies k > \frac{z}{\lg(z + kb)} \geq \frac{kb}{\lg(2kb)} \]

since

\[ g(z) = \frac{z}{\lg(z + kb)} \]
**Bound on VC Dimension**

- **Proof outline** - from last slide:

\[ z \geq kb \implies k > \frac{kb}{\lg(2kb)} \]

But

\[ 2k \leq \frac{9}{10} \sqrt{n} \implies \lg(2 \lg(n)k) < \lg(n) \iff \lg(2bk) < b \]

\[ \implies k < \frac{bk}{\lg(2bk)} \]

So it must be

\[ z < kb \quad \text{i.e.} \quad d - k \lg(n) < k \lg(n) \]

\[ \implies d < 2k \lg(n). \]
Guarantees from Learning Theory

• How does a lower VC dimension theoretically guarantee better learning?

• Let $h =$ the hypothesis chosen by the algorithm

• training_error = percentage of mistakes on training data

• test_error = probability of a mistake on next data point

**Theorem** (Vapnik) With probability at least $1 - \delta$,

$$
test_error \leq training_error + \sqrt{\frac{1}{m} \left( d \log \left( \frac{2em}{d} \right) + \log \left( \frac{4}{\delta} \right) \right)}
$$
Guarantees from Learning Theory

• Can we say anything about the regression case?

• Suppose our training data assumes y-values in the range \([a, b]\).

**Theorem**

With probability \(1 - \delta\),

\[
\text{test\_error} \leq \text{training\_error} + (b - a) \sqrt{\frac{1}{m} \left( d \log \left( \frac{2em}{d} \right) + \log \left( \frac{4}{\delta} \right) \right)}
\]

Error is now based on a regression loss function (such as the L2-norm).
Outline

• Motivation

• Learning Theory
  ✦ Idea of the VC dimension
  ✦ VC dim bounds for sparse classifiers

• Algorithms
  ✦ Complexity
  ✦ Incremental algorithms
Algorithmic Goals

- Columns of $A$ are time series
- If we would like to approximate another time series $b$, try to solve
  \[
  \begin{align*}
  \min & \quad ||x||_0 \\
  \text{s.t.} & \quad Ax \approx b
  \end{align*}
  \]
- To find sparse $x$ with $Ax \approx 0$, we may also solve
  \[
  \begin{align*}
  \min & \quad ||x||_0 \\
  \text{s.t.} & \quad A_i x \approx a_i
  \end{align*}
  \]
  where $a_i$ is the $i^{th}$ column of $A$, and $A_i = A$ with $a_i$ removed
Complexity

• Thus, we may state our ultimate goal as solving the problem

\[
P_0 \begin{cases}
\min ||x||_0 \\
s.t. \quad Ax = b
\end{cases}
\]

• Can we exactly solve this problem?
Complexity

- **Min_Unsatisfy**: given $A, b$, minimize $||Ax-b||_0$
- Arora et al. showed **Min_Unsatisfy** is quasi-NP-hard to approximate within a factor of $e^{\log^{1-\gamma} n}$ for any $\gamma \in (0, 1)$
- **Min_Unsatisfy**$(A, b)$ is equivalent to $P0(C, d)$, where $null(C) = \text{col}(A)$, $d = -Cb$, $y = Ax - b$

**Thus**: problem $P0$ is also quasi-NP-hard to approximate

With Lee-Ad Gottlieb, we have shown **Matrix Sparsification** is also quasi-NP-hard to approximate
Algorithms

• Efficient incremental algorithm to approximate problem P0 over time series

• Can operate on a sliding window or full history

• Can efficiently filter data by a moving geometric taper, if desired

• Achieves running time completely independent of the number of time steps in the sliding window / full history

• Also exists an even faster / slightly less stable version
Algorithms

Three Ideas

• Idea 1: L1-minimization approximates L0-minimization

We want to solve
\[ P_0 \left\{ \begin{array}{l} \min \|x\|_0 \\ s.t. \ Ax = b \end{array} \right. \]

..but it is easier to solve:
\[ P_1 \left\{ \begin{array}{l} \min \|x\|_1 \\ s.t. \ Ax = b \end{array} \right. \]

..and it is often the case that \( P_0(A,b) = P_1(A,b) \).

Note that problem \( P_1 \) may be stated as a linear programming problem.
• Idea ②: The $\varepsilon$-space

One can use the singular value decomposition of matrix $A$:

$$ A = U \Sigma V^T $$

to define a vector space $S$:

$$ x \in S, x \neq 0 \implies \frac{||Ax||}{||x||} \leq \sigma_k $$

If $\sigma_k$ is small, then $x$ is “almost in the null space” in the above sense.
Algorithms

Three Ideas

• Idea ②: The $\varepsilon$-space

How we will use this idea:

If $\sigma_{k+1} < \varepsilon$, and columns of $Q$ are the first $k$ right singular vectors of $A$, then

$$Q^T x = 0 \implies \|Ax\| < \varepsilon\|x\|$$

Therefore: If $\|x\|$ is not too large, then $Ax \approx 0$. 
Algorithms

Three Ideas

• Idea ③ : Need only track the correlation matrix

Recall: our data are time series - the columns of matrix $A$

$A$ is $m \times n$. In many cases, $m \gg n$.

Then $A^T A$ is only $n \times n$, and contains all the information we need:

$$A = U \Sigma V^T \implies A^T A = V \Sigma^2 V^T$$

Very often: Eigenvectors of $A^T A =$ right singular vectors of $A$

**Bonus**: $A^T A$ is easy to update when a single row changes in $A$
Algorithms

- **Main ideas** of the algorithm:

  **Maintain information:**

  \[ B = A^T A \quad \text{and} \quad Q = \text{most\_sig\_eig}(B, k) \]

  **Update:**

  \[
  \begin{align*}
  A & \rightarrow \hat{A} \\
  (\beta X) & \rightarrow (X \alpha)
  \end{align*}
  \]

  \( \beta = \text{old row} \quad \alpha = \text{new row} \)
**Pseudocode**

**Input:** last iteration’s data $B, Q, x$
new row $\alpha$, old row $\beta$

Let $\hat{B} = B - \beta^T \beta + \alpha^T \alpha$

Compute $\hat{Q} = \text{mostSigEig}(\hat{B}, Q, k)$

Solve \[
\begin{cases}
\min \|x\|_1 \\
\text{s.t. } Cx = d
\end{cases}
\]

Where $C = \hat{Q}^T$ without $i^{th}$ column
$d = i^{th}$ column of $\hat{Q}^T$
Time Complexity

- Hard to derive a strong bound due to convergence algorithms used (linear programming and eigenvector computations)

- $O(c k n^2 + \text{LP}_{\text{time}})$
  - $k =$ number of singular vectors tracked
  - $n =$ number of time series
  - $c =$ iterations used in computing eigenvectors of $B$
  - $\text{LP}_{\text{time}} =$ time used by LP solver

- In experiments, one iteration at $n=500$ took about 10 s
Performance Analysis

- $Q$ is $n \times k$, so there is guaranteed to be some $x$ in the null space of $Q^T$ with
  \[ \|x\|_0 \leq k + 1 \]  
  \(\text{sparsity bound}\)

- From $\varepsilon$-space idea,
  \[ Q^T x = 0 \implies \|Ax\| \leq \sigma_{k+1} \|x\|_2 \leq \sigma_{k+1} \|x\|_1, \]  
  and we have minimized \(\|x\|_1\)  
  \(\text{accuracy bound}\)
Experiments

Time (in seconds) per iteration

$n = \text{number of time series}$
Experiments

Relative test error (abs value of) average % of nonzero coeff’s changed per timestep

\( k \approx \text{density of coeff’s} \)

least squares method achieves \( 9.45 \times 10^{-4} \)

sliding window size = 5000

\( k \approx \text{density of coeff’s} \)

\( n = 300 \quad \text{taper} = 0.99 \)

20 iterations per \( k \)
Experiments

Example stock price prediction based on past data

n=500  k=80  window size = 5000
Thanks!

Especially to

Dennis Shasha & Mehryar Mohri

as well as

Lee-Ad Gottlieb
Richard Cole
Sinan Gunturk
Michael Overton
Xiaojian Zhao
Cynthia Rudin
Rosa Figueras

Corinna Cortes
Clifford Hurvich
Nadrian Seeman
Janos Pach
Percy Deift
Daryl Williams
Ashish Rastogi