Random Kernel Perceptron on ATTiny2313 Microcontroller

Nemanja Djuric, Slobodan Vucetic

Department of Computer and Information Sciences
Temple University

SensorKDD, July 25th, 2010, Washington DC
Kernel Perceptron

- Predictor

\[ f(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) \right) \]

- Costs
  - \(O(N)\) space
  - \(O(N)\) update time
  - \(O(N^2)\) total training time

\[ f(\mathbf{x}) = 0 \text{ at time } t = 0 \]

**Inputs**: data sequence \(((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N))\)

**Output**: trained Kernel Perceptron \(f(\mathbf{x})\)

while (training set not empty)

\[
\begin{align*}
  &\text{if } (y_i \cdot f(\mathbf{x}_i) > 0) \\
  &\quad \alpha_i = 0 \\
  &\text{else} \\
  &\quad \alpha_i = y_i \\
\end{align*}
\]

\[ f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \alpha_i \cdot K(\mathbf{x}_i, \mathbf{x}) \]
Random Budget Kernel Perceptron

Idea

- assign support vector budget $T$
- when budget is exceeded, remove a random SV
- resulting predictor

$$f(x) = \text{sign} \left( \sum_{i=1}^{T} \alpha_i K(x,x_i) \right)$$

Costs

- $O(1)$ space
- $O(1)$ update time
- $O(N)$ training time

Inputs : data sequence $((x_1, y_1), ..., (x_N, y_N))$, budget $T$
Output : support vector set $SV = \{SV_i, i = 1 \ldots I\}$

$I \leftarrow 0; i \leftarrow 1$
$SV = \emptyset$
for $i = 1 : N$
{
    if $(y_i \cdot \sum_{j=1}^{I} y_j \cdot K(x_i, x_j) \leq 0)$
    {
        if ($I == T$)
            new = random(I)
        else
            {
                $I \leftarrow I + 1$
                new $\leftarrow I$
            }
    }$SV_{new} = (x_i, y_i)$
}
Motivation

- Random Kernel Perceptron
  - online algorithm
  - low cost
  - easy to implement
  - can solve nonlinear problems
  - accurate

- It still **CANNOT** be implemented on the simplest computers
  - it uses floating-point operations
  - model size easily exceeds available memory

- Goal: Implement Kernel Perceptron on microcontrollers

- Applications
  - sensor networks
  - low-cost online data mining
  - resource-constrained environments
Microcontroller

- ATTiny2313
  - one of the most primitive processors
  - very cheap (< $1)

- Characteristics
  - 128 bytes to store:
    - Kernel Perceptron
    - working variables
  - 2 Kbytes to store program
  - 4 MHz processor speed
  - fixed-point arithmetic (integers)
Some details

- Use Gaussian kernel: \( K(x, x_i) = \exp(\frac{\|x - x_i\|^2}{2^A}) \)

- Resource-saving strategies
  - Quantization of attributes using \( b \) bits
    - trade-off between #SV and #bits
    - quantization loss
  - Approximation of kernel function using only integers and integer calculations
    - we devised an iterative procedure that uses look-up table
    - approximation loss
Results

- Fixed-point vs. floating-point method
- Approximation accuracy (kernel width = $2^A$)
Results

- Accuracy on benchmark datasets

![](image1)

Banana dataset

Checkerboard dataset
Results

- Implementation on microcontroller
- Double-precision Kernel Perceptron: 89.2% accuracy
- Much less memory, faster execution time

<table>
<thead>
<tr>
<th>Banana dataset</th>
<th>4 bits</th>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
<td>Fixed</td>
<td>Float</td>
</tr>
<tr>
<td><em>Data [B] (max 128B)</em></td>
<td>128</td>
<td>379</td>
</tr>
<tr>
<td><em>Program [B] (max 2048B)</em></td>
<td>1720</td>
<td>6012</td>
</tr>
<tr>
<td><strong>Time [ms]</strong></td>
<td>1985</td>
<td>7505</td>
</tr>
<tr>
<td><strong>Accuracy [%]</strong></td>
<td>81.00</td>
<td>81.08</td>
</tr>
<tr>
<td><strong># of SVs</strong></td>
<td>62</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data memory</th>
<th>After quantization</th>
<th>Before (Double-precision)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model</td>
<td>Working variables</td>
</tr>
<tr>
<td>Memory size</td>
<td>70B</td>
<td>58B</td>
</tr>
</tbody>
</table>
Conclusions

- Implemented Kernel Perceptron on ATTiny2313 microcontroller

- Fixed-point calculations of prediction
  - key for implementation
  - low data and program memory
  - speeds up calculations
  - only slightly decreases accuracy

- Our results
  - useful in establishing lower bounds on necessary computational resources for online learning
  - open doors for novel application of data mining, such as data mining from sensor data
Thank you!

- More details in the paper
- Questions? E-mail to nemanja.djuric@temple.edu