Implementation in C+CUDA of Multi-Label Text Categorizers

Lucas Veronese, Alberto F. De Souza, Claudine Badue, Elias Oliveira, Patrick M. Ciarelli
Universidade Federal do Espírito Santo, 29075-910, Vitória-ES
{lucas.veronese, alberto, claudine, elias, pciarelli}@lcad.inf.ufes.br

In automated multi-label text categorization problems with large numbers of labels, the training databases are large, which may render the categorization time prohibitive for online systems. In this work, we evaluate the parallel implementation in C+CUDA of two multi-label text categorizers: the first is based on the k-Nearest Neighbors (k-NN) algorithm [1] and the second is based on Probabilistic Neural Networks (PNN) [2]. We implemented these algorithms in three different ways: sequential in C, parallel in C+CUDA and parallel using the C+CUBLAS library.

The k-NN categorizer finds the k nearest neighbors of an input document \( d_i \) in the set of previously learned documents, \( TV \), according to some given distance metric —in the experiments reported in this paper, we used the cosine of the angle between the floating-point vector that represents the input document \( d_i \) (bag-of-words document representation [1]) and each document \( d \in TV \), is given by:

\[
\cos(d_i, d) = \frac{d_i \cdot d}{|d_i||d|} \tag{1}
\]

The k-NN categorizer employs a function \( f(d_i, c) \) that returns the highest value of \( \cos(d_i, d) \), for \( d \in TV \) and \( c \in C \), where \( C \) is the set of pertinent categories for the document \( d_i \). It selects the \( k \) pairs \( (d, c) \in (D \times C) \) from the top of the ranking derived from \( f(c) \).

The PNN used in this work was proposed by Oliveira et. al [2] and is composed of two feed-forward layers: pattern layer and summation layer. In the training phase, for each document \( d_i \) is created a set of neurons, one for each category \( c_j \in C \), where each neuron \( n_i \) stores the vector \( d_i \) as a vector of term weights, \( w_{ik} \). In the categorization phase, an input document \( d_i \) is presented to the pattern layer. The \( i \)-th neuron, \( n_i \), associated to category \( c_j \) in the pattern layer, calculates the activation function \( A(d_i, c_j, n_j) \) for document \( d_i \), given by:

\[
A(d_i, c_j, n_j) = \frac{1}{2\pi\sigma} \exp\left(-\frac{d_i^2 - 2d_i c_j + c_j^2}{\sigma^2}\right), k=1, \ldots, |C|, i=1, \ldots, |D| \tag{2}
\]

where \( \sigma \) is a constant for all neurons (adjusted during training for best categorization performance [2]), \( C \) is the whole set of possible categories, and \( D \) is the set of documents associated to category \( c_j \). In the summation layer, which has as many neurons as \( |C| \), each neuron is associated with a category \( c_k \) and computes the function \( f(d_i, c_k) \):

\[
f(d_i, c_k) = \sum_{k=1}^{N_k} A(d_i, c_k, n_k), k=1, \ldots, |C| \tag{3}
\]

where \( N_k \) is the number of neurons of the pattern layer associated to \( c_k \). The categories \( c_k \) ranked above a threshold are predicted to the input document \( d_i \).

We ran the C, C+CUDA and C+CUBLAS versions of our categorizers in an AMD Athlon X2 (Dual Core) 5,200+ of 2.7 GHz, with 3GB of 800 MHz DRAM DDR2, and video card NVIDIA GeForce GTX 285, with 1GB of DRAM GDDR3.

The data set used is composed of 6,911 documents categorized into 105 different categories by specialists in the domain of the documents. Each one of these categories occurs in exactly 100 different documents, i.e., there are 100 documents of each category. Each document is represented by a vector of single precision floats of size 3,764 (the number of relevant terms in the system vocabulary).

To evaluate the performance of our categorizers in terms of time, we selected 6,910 documents of the data set for training, and a single one for testing the categorizers. Each categorizer was executed 100 times and the average was used to compare them. Table 1 shows the average times for each categorizer (rows) and categorizer implementation (columns), in addition to the speed-ups over the sequential implementation (last two columns). As the table shows, we achieved speed-ups of about 60 for the C+CUDA version and about 45 for the C+CUBLAS version. These results show that, with CUDA, it is possible to implement online text categorization and that, in some cases, it is worth implementing the whole code instead of using C+CUBLAS.

Table 1: Average times and speed-ups of our text categorizers

<table>
<thead>
<tr>
<th>Categorizer</th>
<th>C</th>
<th>C+CUDA</th>
<th>C+CUBLAS</th>
<th>Speed-up C+CUDA</th>
<th>Speed-up C+CUBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-NN</td>
<td>0.1928</td>
<td>0.0030</td>
<td>0.0042</td>
<td>64.26</td>
<td>44.90</td>
</tr>
<tr>
<td>PNN</td>
<td>0.1938</td>
<td>0.0033</td>
<td>0.0044</td>
<td>58.72</td>
<td>44.04</td>
</tr>
</tbody>
</table>