Number 107 October 2016

ERCIM NEWS

www.ercim.eu

Special theme:

Machine Learning

Also in this issue:

Research and Society:

Open Access
Open Science

ERCIM News is the magazine of ERCIM. Published quarterly, it reports on joint actions of the ERCIM partners, and aims to reflect the contribution made by ERCIM to the European Community in Information Technology and Applied Mathematics. Through short articles and news items, it provides a forum for the exchange of information between the institutes and also with the wider scientific community. This issue has a circulation of about 6,000 printed copies and is also available online.

ERCIM News is published by ERCIM EEIG
BP 93, F-06902 Sophia Antipolis Cedex, France
Tel: +33 4 9238 5010, E-mail: contact@ercim.eu
Director: Jérôme Chailloux, ISSN 0926-4981

Contributions
Contributions should be submitted to the local editor of your country

Advertising
For current advertising rates and conditions, see http://ercim-news.ercim.eu/ or contact peter.kunz@ercim.eu

ERCIM News online edition
http://ercim-news.ercim.eu/

Next issue
January 2017, Special theme: Computational Imaging

Subscription
Subscribe to ERCIM News by sending an email to en-subscriptions@ercim.eu or by filling out the form at the ERCIM News website: http://ercim-news.ercim.eu/

RESEARCH AND SOCIETY

The section "Research and Society" on "Open Access – Open Science" has been coordinated by Laurent Romary (Inria)

SPECIAL THEME

The special theme section "Machine Learnig" has been coordinated by Sander Bohte (CWI) and Hung Son Nguyen (University of Warsaw)

RESEARCH AND INNOVATION

This section features news about research activities and innovative developments from European research institutes

EVENTS, IN BRIEF

Fast Traversal of Large Ensembles of Regression Trees

by Claudio Lucchese, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto (ISTI-CNR), Salvatore Orlando (Ca' Foscari University of Venice) and Rossano Venturini (University of Pisa)

The complexity of tree-based, machine-learnt models and their widespread use in web-scale systems requires novel algorithmic solutions to make the models fast and scalable, both in the learning phase and in the real-world.

Machine-learnt models based on additive ensembles of regression trees have been shown to be very effective in several classification, regression, and ranking tasks. These ensemble models, generated by boosting meta-algorithms that iteratively learn and combine thousands of simple decision trees, are very demanding from a computational point of view. In fact, all the trees of the ensemble have to be traversed for each item to which the model is applied in order to compute their additive contribution to the final score.

This high computational cost becomes a challenging issue in the case of large-scale applications. Consider, for example, the problem of ranking query results in a web-scale information retrieval system: the time budget available to rank the possibly huge number of candidate results is limited due to the incoming rate of queries and user expectations of quality-of-service. On the other hand, effective and complex rankers with thousands of trees have to be exploited to return precise and accurate results [1].

To improve the efficiency of these systems, in collaboration with Tiscali Italia S.p.A, we recently proposed QuickScorer (QS), a solution that remarkably improves the performance of the scoring process by dealing with features and characteristics of modern CPUs and memory hierarchies [2]. QS adopts a novel bit-vector representation of the tree-based model, and performs the traversal of the ensemble by means of simple logical bitwise operations. The traversal is not performed by QS one tree after another, as one would expect, but is instead interleaved, feature by feature, over the whole tree ensemble. Due to its cache-aware approach, both in terms of data layout and access patterns, and to a control flow that entails very low branch misprediction rates, the QS performance is impressive, resulting in speedups of up to 6.5x over state-of-the-art competitors.

An ensemble model includes thousands of binary decision trees, each composed of a set of internal nodes and a set of leaves. Each item to be scored is in turn represented by a real-valued vector x of features. As shown in Figure 1, the internal nodes of all the trees in the ensemble are associated with a Boolean test over a specific feature of the input vector (e.g., $x[4] \leq \gamma 2$). Each leaf node stores the potential contribution of the specific tree to the final score of the item. The scoring process of each item requires the traversing of all the trees in the ensemble, starting at their root nodes, until a leaf node is reached, where the value of the prediction is considered. Once all the trees in the ensemble have been visited, the final score for the item is given by the sum of the partial contributions of all the trees.

One important result of QS is that to compute the final score, we only need to identify, in any order, all the internal nodes of the tree ensemble for which the Boolean tests fail, hereinafter false nodes. To perform this task efficiently, QS relies on a bit-vector representation of the trees. Each node is represented by a compact binary mask identifying the leaves of the current tree that are unreachable when the corresponding node test evaluates to false. Whenever a false node is found, the set of unreachable leaves, represented as a bit-vector, is updated through a logical AND bitwise operation. Eventually, the position of the leaf storing the correct contribution for each tree is identified. Moreover, in order to find all the false

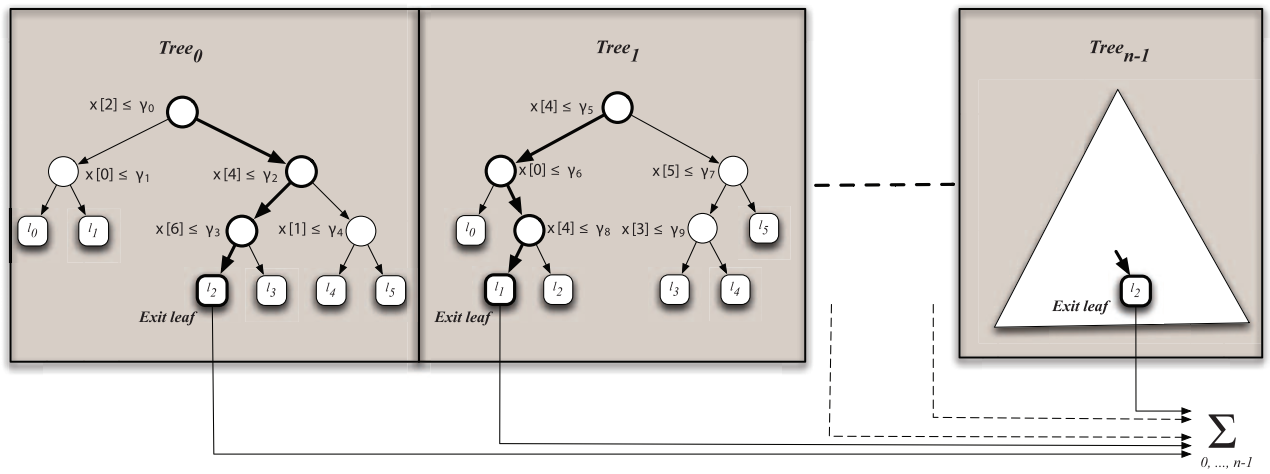

Figure 1: An ensemble of binary decision trees.

nodes for the scored item efficiently, QS processes the nodes of all the trees feature by feature. Specifically, for each feature *x[i]*, QS builds the list of all the nodes of the ensemble where *x[i]* is tested, and sorts this list in ascending order of the associated threshold γk. During the scoring process for feature *x[i]*, as soon as the first test in the list evaluating to true is encountered, i.e., $x[i] \leq \gamma k$, the subsequent tests also evaluate to true, and their evaluation can be safely skipped and the next feature *x[i+1]* considered.

This organisation allows QS to actually visit a consistently lower number of nodes than in traditional methods, which recursively visit the small and unbalanced trees of the ensemble from the root to the exit leaf. In addition, QS exploits only linear arrays to store the tree ensemble and mostly performs cache-friendly access patterns to these data structures.

Considering that in most application scenarios the same tree-based model is applied to a multitude of items, we recently introduced further optimisations in QS. In particular, we introduced vQS [3], a parallelised version of QS that exploits the SIMD capabilities of mainstream CPUs to score multiple items in parallel. Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) are sets of instructions exploiting wide registers of 128 and 256 bits that allow parallel operations to be performed on simple data types. Using SSE and AVX, vQS can process up to eight items in parallel, resulting in a further performance improvement up to a factor of 2.4x over QS. In the same line of research we are finalising the porting of QS to GPUs, which, preliminary tests indicate, allows impressive speedups to be achieved.

More information on QS and vQS can be found in [2] and [3].

References:
[1] G. Capannini, et al.: "Quality versus efficiency in document scoring with learning-to-rank models", Information Processing & Management, Elsevier, 2016, http://dx.doi.org/10.1016/j.ipm.2016.05.004.
[2] C. Lucchese et al.: "QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees", ACM SIGIR 2015: 73-82 [best paper award].
[3] Cl. Lucchese, et al.: "Exploiting CPU SIMD Extensions to Speed-up Document Scoring with Tree Ensembles", ACM SIGIR 2016: 833-836.

Please contact:
Raffaele Perego
ISTI-CNR, Pisa, Italy
+39 (0)50 3152993
raffaele.perego@isti.cnr.it

Optimising Deep Learning for Infinite Applications in Text Analytics

by Mark Cieliebak (Zurich University of Applied Sciences)

Deep Neural Networks (DNN) can achieve excellent results in text analytics tasks such as sentiment analysis, topic detection and entity extraction. In many cases they even come close to human performance. To achieve this, however, they are highly-optimised for one specific task, and a huge amount of human effort is usually needed to design a DNN for a new task. With DeepText, we will develop a software pipeline that can solve arbitrary text analytics tasks with DNNs with minimal human input.

Assume you want to build a software for automatic sentiment analysis: given a text such as a Twitter message, the tool should decide whether the text is positive, negative, or neutral. Until recently, typical solutions used a feature-based approach with classical machine learning algorithms (e.g., SVMs). Typical features were number of positive/negative words, n-grams, text length, negation words, part-of-speech tags etc. Over the last two decades a huge amount of research has been invested in designing and optimising these features, and new features had to be developed for each new task.

With the advent of deep learning, the situation has changed: now the computer is able to learn relevant features from the texts by itself, given enough training data. Solving a task like sentiment analysis now requires three major steps: define the architecture of the deep neural network; aggregate enough training data (labelled and unlabelled); and train and optimise the parameters of the network.

For instance, Figure 1 shows the architecture of a system that won Task 4 of SemEval 2016, an international competition for sentiment analysis on Twitter [1]. This system uses a combination of established techniques in deep learning: word embedding and convolutional neural networks. Its success is primarily based on three factors: a proper architecture, a huge amount of training data (literally billions of tweets), and a huge amount of computational power to optimise its parameters. Live demos of various deep learning technologies are available at [2].

Goal of DeepText

In DeepText, we will automate the three steps above as far as possible. The ultimate goal is a software pipeline that works as follows (see Figure 2):

1. The user uploads his or her training data in a standard format. The data can consist of unlabelled texts (for pre-training) and labelled texts, and the labels implicitly define the task to solve.
2. The system defines several DNNs to solve the task. Here, different fundamental architectures will be used, such as convolutional or recurrent neural networks.
3. The system then trains these DNNs and optimises their parameters.