Article

Improvement in the Efficiency of a Distributed Multi-Label Text Classification Algorithm Using Infrastructure and Task-Related Data

Martin Sarnovsky * and Marek Olejnik
Department of Cybernetics and Artificial Intelligence, Technical University Košice, Letná 9/A, 040 01 Košice, Slovakia; marekolejnik17@gmail.com
* Correspondence: martin.sarnovsky@tuke.sk

Received: 2 January 2019; Accepted: 9 March 2019; Published: 18 March 2019

Abstract: Distributed computing technologies allow a wide variety of tasks that use large amounts of data to be solved. Various paradigms and technologies are already widely used, but many of them are lacking when it comes to the optimization of resource usage. The aim of this paper is to present the optimization methods used to increase the efficiency of distributed implementations of a text-mining model utilizing information about the text-mining task extracted from the data and information about the current state of the distributed environment obtained from a computational node, and to improve the distribution of the task on the distributed infrastructure. Two optimization solutions are developed and implemented, both based on the prediction of the expected task duration on the existing infrastructure. The solutions are experimentally evaluated in a scenario where a distributed tree-based multi-label classifier is built based on two standard text data collections.

Keywords: text classification; multi-label classification; distributed text-mining; task assignment; resource optimization; grid computing

1. Introduction

Knowledge discovery in texts (KDT), often referred to as text-mining, is a special kind of knowledge discovery in databases (KDD) process. It is usually complex because of the processing of unstructured data and covers the topic of natural language processing [1]. The overall structure of the process follows typical KDD patterns, and the main difference is in the transformation of the unstructured text data into a structured format ready to be used by data mining algorithms.

A KDT process usually consists of several phases that can be mapped to standards for KDD, such as CRISP-DM (CRoss-Industry Standard Process for Data Mining) [2,3]. The data and domain understanding phase identify the most important concepts related to the application domain. The main objective of this phase is also to specify the overall goal of the process. The data gathering phase is aimed at the selection of textual documents for the solution of a specified task. It is important to select the documents that cover the application domain or may be related to the task solution. Various techniques can be applied, including manual or automatic selection of the documents or leveraging of the existing corpora. The preprocessing phase involves the application of the operations that transform the textual data into the structured format. Usually, the dataset is converted into one of the commonly used representations (e.g., vector space model). During this phase, text preprocessing techniques are also applied. These techniques are often language-dependent and include stop word removal, stemming, lemmatization, and indexation. The text-mining phase represents the application of the model-building algorithm. Depending on the task, classification, clustering, information retrieval, or information extraction models are created. The models are evaluated in the next step, and then the results can be visualized and interpreted, or models can be applied to the real environment.
The classification of text documents is one of the specific text-mining tasks, and its main goal is to assign the document to one of the pre-defined classes (categories). Usually, the classifier is built on a set of labeled data (train data). A trained model is evaluated on a test dataset and then can be deployed to be used on live, unlabeled data. There are various classification methods available, including tree-based classifiers, neural networks, support vector machines, k-nearest neighbors, etc.

In text classification tasks, we often deal with a multi-class classification problem (e.g., a classification task where a document may be assigned to more than one category) [4]. Certain types of classifiers can handle multi-class data and can be directly used to solve multi-class classification problems (e.g., probabilistic classifiers). In order to use other types of classifiers, training algorithms have to be adapted. One of the commonly used ways of adapting an algorithm to handle multi-class data is to build a set of binary classifiers of a given model, each dedicated to a particular category. The final model then consists of a set of these binary classifiers.

When training models on real text collections, we often face the problem of processing large volumes of data, which requires the adoption of advanced technologies for distributed computing. To solve such computationally intensive tasks, training algorithms are modified to leverage the advantages of distributed computing architectures and implemented using different programming paradigms and underlying technologies.

However, the distributed implementation itself cannot guarantee the effective usage of the available computing resources, especially in environments with limited infrastructures. Therefore, various techniques for the improvement of resource allocation and the optimization of infrastructure usage have been developed. In some tasks, however, resource allocation and task assignment can be heavily influenced by the task itself and its parameters. In this paper, we propose a method that can optimize the effectiveness of resource usage in the distributed environment as well as a task assignment process specifically for the tasks associated with building distributed text classification models.

The paper is organized as follows: First, we give an overview of the distributed text classification methods and tools and introduce the algorithms used in this paper. The next section describes the pre-existing optimization techniques used in similar tasks. The section that follows presents the description of the developed and implemented methods. Finally, the results of the experimental evaluation are discussed in Section 6 and summarized in Section 7.

2. Distributed Classification

In general, there are two major approaches to the distribution of KDD and KDT model building algorithms:

- **Data-driven decomposition**—in this case, we assume that the decomposition of the dataset is sufficient. Input dataset $D$ is divided into $n$ mutually disjoint subsets $D_n$. Each of the subsets then serves as a training set for the training of partial models. When using this approach, a merging mechanism aggregating all of the partial models has to be developed. There are several models suitable for this type of decomposition, such as k-nearest neighbors (k-NN), support vector machine classifier, or all instance-based methods in general.

- **Model-driven decomposition**—in this case, the input dataset remains complete and the algorithm is modified to run in a parallel or distributed way. In general, it is the decomposition of the model building itself. The nature of the decomposition is model-specific, and we can generally state that the partial subprocesses of partial model building have to be independent of each other. Various models can be decomposed in this way, such as tree-based classifiers, compound methods (boosting, bagging), or clustering algorithms based on self-organizing maps.

According to [5], it is more suitable to use a more complex algorithm on a single node applied to a subset of the dataset. However, the dataset splitting and distribution of the data subsets to the nodes can be more time-consuming. This approach is also not suitable for data mining tasks from raw data, where their integration and preprocessing are needed, as it requires more transfers of the data between
the nodes. The second approach is more suitable when using large unstructured datasets (e.g., text data), but it is more complex to design the distributed algorithm itself. Often, the communication cost for constructing a model is rather high.

If the dataset is represented as a set of $n$-tuples, where each tuple represents particular attribute values, there are two approaches to data fragmentation [6]:

- **Horizontal fragmentation**—data are distributed in such a way that each node receives a part of the dataset, in the case that the dataset comprises $n$-tuples, each node receives a subset of $n$-tuples.
- **Vertical fragmentation**—in this case, partial tuples of a complete dataset are assigned to the nodes.

There are various existing implementations of distributed and parallel data as well as text mining models using various underlying technologies and tools. Several machine learning libraries are available offering algorithm implementations in MapReduce (e.g., Mahout) [7], on top of the hybrid processing platforms such as Spark (MLlib, ML Pipelines, city, state abbreviation if USA, country) [8] or a number of specific algorithm implementations using grid computing or MPI (message parsing interface). In [9] authors describe PLANET, a distributed, scalable framework for the classification of trees, building on large datasets. The tree-building algorithm is implemented using the MapReduce paradigm and aims to maximize the number of nodes that can be expanded in parallel, while considering memory limitations. It also aims to store in memory all the assigned training data partitions on particular nodes. Another implementation based on Apache Spark uses similar techniques such as Hadoop MapReduce implementations [10], while several other works leverage the computational power of GPUs (Graphics Processing Units) to improve the performance of the MapReduce implementations. Caragea et al. [11] describe a multi-agent approach to building tree-based classifiers. Their approach is aimed at building models in distributed datasets and minimizing communication between the nodes in distributed environments. When applied in the realm of big data, there are numerous approaches that have already been published. In the area of multi-label distributed classification algorithms, studies have presented classifiers able to handle data with hundreds of thousands of labels [12], and more recent work can be found in References [13–16]. However, those approaches focus mostly on handling extremely large sets of target labels.

In our work, we used our own implementations of classification and clustering algorithms in the Java Bag of Words library (Jbowl) [17]. Jbowl provides an API (Application Programming Interface) for building text mining applications in Java and contains various tools for text preprocessing, text classification, clustering, and model evaluation techniques. We designed and implemented distributed versions of classification and clustering models from the Jbowl library. The GridGain platform [18] was used as a distributed computing framework. Table 1 summarizes the currently implemented sequential and distributed versions of the algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision tree classifier</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>k-Nearest neighbor classifier</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Rule-based classifier</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Support vector machine classifier</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Boosting compound classifier</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>k-Means clustering</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GHSOM clustering</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

The implementation of the distributed tree-based classifier addresses the multi-label classification problem (often present in text classification tasks). Each class is considered as a separate binary classification problem, and the final classifier consists of a set of binary classifiers for each particular class/category. In our implementation, binary classifiers were built in a distributed way.
The traditional approach to single-label classification is based on the training of a model, where each training document is assigned to one of the classes. In the case of multi-label classification, documents are assigned to more classes at the same time (e.g., a news document may belong to both “domestic news” and “politics”). Authors in [4] describe how traditional methods can be applied to solve the multi-class problem:

- **Problem transforming methods**—methods that adapt the problem itself, for example, transforming the multi-label classification problem into a set of binary classification problems.
- **Algorithm adaptation**—approaches that adapt the model itself to be able to handle multi-class data.

The distributed induction of decision trees is one of the possible ways to reduce the time required to build a classifier on the large data collections. From the perspective of the model or data-driven paradigm, parallelism can be achieved by the parallel expansion of decision tree nodes (model-driven parallelization) or by the distribution of a dataset and building partial models on these partitions (data-driven parallelization). According to [23], building distributed tree models is a complex task. One of the reasons for this is that the structure of the final tree is often irregular, which places different requirements on the computational capabilities of the nodes responsible for the expansion of particular nodes. This can lead to an increase in the total time taken to build a model. A static scheme for task allocation can prove to be unsuitable when applied to unbalanced data. Another reason is that even if the nodes can be expanded in parallel, all training set data shared from the tree nodes at the same level are still required for model building. There are several strategies for implementing distributed tree model building; several other options also exist in the case of multi-label tree models. The process of building a classifier corresponds to particular CRISP-DM phases and can be summarized as follows (see Figure 1):

- **Data preparation**—a selection of textual documents, and selection of a training and testing set.
- **Data preprocessing**—at the beginning of the process, preprocessing of the complete dataset is performed. This includes text tokenization, lowercase transformation, and stop word removal. Then, a vector representation of the textual documents is computed using tf-idf (term frequency-inverse document frequency) weighting.
- **Model building**—in this step, a tree-based classifier is trained on the training set. The result is a classification model ready to be evaluated and deployed.

![Figure 1. Sequential model building.](image)

A distributed tree-based classifier is trained in a similar way. The biggest difference is in the process of processing a vector space text model and model building. In this case, the text model building is divided into sub-tasks guided by a master node in the distributed infrastructure (see Figure 2). The master node assigns the sub-tasks to the worker nodes (with assigned data). When the task
assignment is complete, partial models are created on the available computing nodes. When all assigned sub-models are created, the computing node sends partial models to the master node. When all computational nodes deliver partial models to the master node, they are collected and merged into the final classification model.

![Distributed model building](image)

**Figure 2.** Distributed model building.

In the task assignment step, optimization methods can be applied. This paper presents two solutions to the sub-task allocation to available computational grid nodes, and both methods are based on the estimation of the expected time required to build partial models on the node. Several parameters must be taken into consideration when estimating the build time, including sub-task complexity, overall task parameters (in text classification these are the number of terms, or categories), or the computational power of the available nodes. The expected time of the sub-task building is influenced by two task parameters: document frequency (i.e., number of documents in a particular category) and the number of terms in documents from that category. The function dependency between the task time and those parameters were estimated using a model built on the data from the previous experiments in the grid environment. The following sections give a description of the presented methods.

### 3. Optimization of the Classifier Building Using Dataset- and Infrastructure-Related Data

**Optimization of Task Assignment in Distributed Environments**

There are various studies presenting a wide range of methods used to solve the task assignment problem in distributed environments. In [24] authors describe the scheduling of tasks in the grid environment using ant colony optimization, [25] presents the task assignment problem solved by means of a bee colony algorithm, and authors in [26] present the same problem solved by using directed graphs. Dynamic resource allocation aspects are addressed in [27]. The MapReduce framework solves the unbalanced sub-task distribution by running local aggregations (local reducers) on the mappers that can prepare the mapper intermediate results for the reduce step. However, the mapping phase has to be finished completely in order to start the reducer phase, so unbalanced assignment can lead to uneven node utilization and prolong the overall processing time.

In some specific tasks, the sub-task complexity is related to factors other than the data size. On the other hand, performance parameters and utilization of the available nodes can affect the overall task processing. In [28] some of the issues related to MapReduce performance in distributed environments are addressed in heterogeneous clusters, with authors focusing on the unreasonable allocation of tasks to the nodes with different computational capabilities to prove that such optimization brings significant benefits and greatly improves the efficiency of MapReduce-based algorithms. Authors in [29] solve the problem of task assignment and resource allocation in distributed systems using a genetic algorithm. Optimal assignment minimizes the costs of task processing with respect to the specified constraints (e.g., available computing nodes, etc.). The costs of sub-task processing on a particular node are represented by the cost matrix and, in a similar fashion, the communications
matrix stores the costs of communication between the nodes during the creation of sub-tasks. A genetic algorithm is then used to minimize the cost of the allocation function.

The main objective of the work presented in this paper is to develop and evaluate a method for the improvement of task assignment to a particular set of specific text mining tasks—the building of multi-label classification models in the grid environment. We decided to combine several aspects in order to leverage both the performance-related information obtained from the distributed infrastructure and the task-related data extracted from the dataset and improve the task assignment using these data by solving the assignment problem. Especially when building classification models on large text corpora, these methods can bring significant benefits in terms of resource utilization.

4. Design and Implementation of Optimization Mechanisms

In some cases, text mining processes are rather complex and often resource-intensive. Therefore, solving a text mining task (as an iterative and interactive process) can consume substantial time and computational resources. Our main aim was to extend the existing techniques for text mining model building with optimization methods in order to improve resource utilization of the distributed platform. In general, our focus was on gathering all the relevant data from the platform as well as data related to the task itself and leverage that information for the improvement of the resource effectiveness of the implemented distributed algorithms.

We used data extracted from the dataset, such as the size of the processed data, the structure of the data (e.g., for classification tasks, we used the number of classes in the training set, distribution of documents among the classes, etc.). We also used data gathered from the distributed infrastructure. Those were used to describe the actual state of the platform, actual state of the particular computing nodes, their performance, and available capacity. We identified the most relevant data, which we used in both solutions described in the following sections:

- Task-related data
  - Dataset characteristics
    - Number of documents in a dataset;
    - Number of terms in particular documents;
  - The frequency of category occurrence (in classification tasks)—one of the most important criteria influencing the complexity of partial models (the most frequent categories result in the most complex partial models).

- Infrastructure-related data
  - Number of available computing nodes;
  - Node parameters
    - Number of CPU cores;
    - Available CPU;
    - Available RAM;
    - Heap space.

The following sections present the designed and implemented optimization mechanisms based on the above-mentioned data.

4.1. Tasks Assignment with No Optimization

The first optimization solution is based on the assignment of sub-tasks to grid computational nodes according to the task and infrastructure data. No optimization method is used in this case.
The first step is the initialization of the node and task parameters.

- The variable describing the overall node performance is also initialized.
- The algorithm checks the available grid nodes, and the values of the actual node parameters of all grid nodes are set.
- When the algorithm finishes checking all the available nodes, it checks the maximum value of the obtained parameter values among the grid nodes (for each parameter). Then, all node parameter values are normalized (to \( <0,1> \) interval).
- In the next step, an actual node performance parameter is computed as the sum of all parameter values. It is possible to set the weights of each parameter when a certain resource is more significant (e.g., the task is more memory- or CPU-intensive). In our case, we used equal weight values. Nodes are then ordered by the assigned performance parameters and an average node (with average performance parameters) is found.
- The next step computes the maximum number of sub-tasks assigned to a given node. A map is created storing statistics describing the sub-tasks' complexity information extracted from the task-related data. Sub-tasks (binary classifiers) are ordered according to the frequency parameter.
- Then, the average document frequency of a binary classifier is computed. This represents the maximum number of sub-tasks that can be assigned to computational nodes with an average overall performance. For the more powerful nodes, the limit is increased, and it is decreased for the less powerful ones. The increase/decrease is computed in the same ratio as the performance parameters ratio between a particular and average node.
- Each available node is then equipped with a specific number of assigned sub-tasks in the same way as in the non-optimized distributed model. The number of assigned tasks can be exceeded in the final assignment in some cases (e.g., in a situation where all sub-tasks could not fit into the computed assignments).

This method is rather simple and serves as the basis for the method using the optimization task assignment problem described further.

### 4.2. Task Assignment Using Assignment Problem

The initial phase of the second proposed solution is the same as in the first approach. The difference is that the particular sub-task assignment is solved as a combinatorial optimization problem (assignment problem). As a special type of transportation problem, the assignment problem is specified by a number of agents and a number of tasks [30]. Agents can be assigned to perform a specific task, and this activity is represented by a cost. This cost can vary depending on the task assignment. The goal is to perform all tasks by assigning exactly one agent to each task in such a way that the total cost of the assignment is minimized [31]. In this particular case, we solved the generalized assignment problem, where \( m \) tasks had to be assigned to \( n \) available computational nodes. The goal was to perform the assignment to minimize the optimization function, which in this case represented the computational time of the task. In distributed model building, the overall task completion time is heavily influenced by the completion time of the longest sub-task. Therefore, we decided to specify the constraints to ensure the even distribution of sub-tasks among the available nodes.

In our approach, we compute matrix \( M \), where \( m_{ij} \) (where \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \) represents the predicted times (obtained in the same way as in the first presented solution) of task \( i \) on computational node \( j \) serves as input data for the assignment task. Each node is also graded based on its computational power, in the same way as in the first solution.

The assignment task is solved under two sets of constraints. The first constraint specifies that each task can be assigned to one particular available node:

\[
\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, \ldots, m.
\]
The second constraint ensures that the task distribution to the nodes is homogeneous; that is, each node is assigned a number of sub-tasks which should (when taking into consideration the computational power of the nodes) take the minimum overall computation time. This is specified by the criterion function:

\[ \sum_{i=1}^{m} \sum_{j=1}^{n} m_{i,j} c_j x_{i,j} = \text{MIN}, \]

where \( m_{i,j} \) are estimated times of task \( i \) on node \( j \), \( c_j \) represents the coefficient of computational power of node \( j \) and \( x_{i,j} = [0, 1] \) where \( x_{i,j} = 1 \) when task \( i \) is assigned to node \( j \), otherwise \( x_{i,j} = 0 \). A set of constraints specifies the homogeneous distribution of tasks among the nodes:

\[ \sum_{i=1}^{m} m_{i,j} c_j x_{i,j} = \frac{\sum_{i=1}^{m} m_{i,\text{avg}}}{n} \times k, \forall j = 1, \ldots, n, \]

where \( m_{i,\text{avg}} \) is the total task completion time on an average node (computed from all nodes in the grid) and \( k = 1 \) is the tuning parameter. When the algorithm is not able to find a solution, the parameter \( k \) is increased by 0.1 until a solution is found. Once the task assignment is completed, the algorithm continues similarly as in the previous solution and distributes sub-tasks to the assigned nodes, builds the sub-models, and merges them into the final model. We used the IPOPT (Interior Point OPTimizer) [32] solver implemented in the JOM (Java Optimization Modeler—a Java-based open-source library for solving optimization problems) library to solve the assignment problem. IPOPT is designed to cover a rather broad range of optimization problems. The results sometimes rely on picking up a starting point, which can result in the solver becoming stuck in a local optimum. To remove this limitation, it is possible to restart the optimizer with a perturbed found solution and resolve the optimizer again.

5. Experiments

The experiments were performed in the testing environment comprising 10 workstations connected via the local 1 Gbit intranet. Each connected node was equipped with an Intel Xeon Processor W3550 clocked at 3.07 GHz CPU, 8 GB RAM, and 450 GB of available storage. We used the multi-label tree-based classifier implementation described in [33]. The algorithm was implemented in Java using the Jbowl library for specific text-processing methods and using GridGain as the platform for distributed computing.

The main purpose of the conducted experiments was to compare both of the designed task distribution algorithms in the task of building classification. Our main goal was to compare the sub-task distribution in both proposed solutions and to compare them with the distributed model building without implementing optimization. We focused on particular nodes load and the distribution of particular sub-tasks to the computing nodes, and measured the time needed to complete the sub-tasks as well as the whole job. A specific set of experiments was conducted to prove how the task balancing algorithms dealt with the heterogeneous environment. The experiments were performed using two standard datasets in the text classification area: the Reuters 21,578 dataset (ModApte split) and a subset of MEDLINE corpus. Both datasets represent the traditional standard data collections frequently used for benchmarking of the text classification algorithms. Both can be used to demonstrate the proof of concept of the presented approach. Figures 3 and 4 show the structure of the dataset in terms of the category frequency distribution, giving an overview of the distribution of sub-task complexity.
The first round of experiments was aimed at comparing different approaches to task distribution when building a multi-label tree-based classifier in a homogeneously distributed environment. We compared the proposed solutions with the distributed multi-label algorithm with no optimization and with a very simple document frequency-based criterion. We also compared the completion time of the model building. The experiments were conducted on a grid consisting of 2, 4, 6, 8, and 10 computational nodes, each of about the same configuration. Figures 5 and 6 give the experimental results on both datasets and show that the proposed optimization methods may reduce the overall model construction time, even if in a balanced infrastructure with the homogeneous environment and equally powerful computational nodes. In both experiments, the addition of more than 10 nodes did not bring any benefit to the task completion. This was mainly caused by the structure of the datasets, as the minimum task completion time was represented by the completion time of the most complex sub-task. Each of the implemented optimization methods came close to that limitation.
The most significant improvements were noticed mostly in environments with fewer computational nodes. The performance of particular computational nodes was also evaluated.

Our main intention was to investigate how the distribution was performed and how the sub-tasks were assigned to the computational nodes. Figures 7 and 8 give the performance of nodes on both datasets in the homogeneous environment, summarize the completion times of the sub-tasks, and show how the nodes were utilized during the overall process of model building.
The second round of experiments was conducted in a heterogeneous environment. The computational power of the nodes was different, as we altered two nodes in a four-node experiment. The configuration of Node 1 was significantly improved (more CPUs and RAM). We also simulated Node 3 loaded with other applications or processes, so the available CPU and RAM parameters were significantly lower. We conducted a set of experiments on the configuration of four nodes on both datasets, and similarly to the previous experiment, we focused on the performance of the nodes and the completion time of the assigned sub-tasks. The results of both solutions on the Reuters dataset are given in Figure 9. Figure 10 shows the same experimental results obtained on the MEDLINE data.
Another factor could also be the fact that most text classification tasks are multi-label problems. This can also be a factor when decomposing the model building just by splitting the data. From this perspective, multi-label text classification serves as a good example to evaluate optimization methods based on task and environment data.

6. Discussion

Experiments performed on the selected dataset proved the usability of the designed solutions for the optimization of sub-task distribution based on the actual state of the used computing infrastructure and task-related data. Various approaches for distributed machine learning algorithms are available (see Section 2), but most of them are mostly based on the data-based distribution paradigm. Most large-scale distributed models (e.g., those based on the MapReduce paradigm) divide the training data into sub-sets to train the partial models in a distributed fashion. In some cases, this decomposition can lead to an unbalanced distribution of workload. This can be specific for the text processing domain, as textual documents are usually in different sizes and contain a variable number of lexical units. Another factor could also be the fact that most text classification tasks are multi-label problems. This can also be a factor when decomposing the model building just by splitting the data. From this perspective, multi-label text classification serves as a good example to evaluate optimization methods based on task and environment data.

Task-related data used for optimization are strictly tied to the solved problem and underlying processed data. In our case, we selected factors that greatly influence the computing resources requirements in the model building phase. Dataset characteristics can be obtained directly from the data, and these factors could also be considered in different text-processing problems (e.g., clustering). Similar factors could be identified for standard structured data-mining problems (instead of the number of terms, the number of attributes could be used as well as their respective statistics). In our approach we used a category occurrence frequency, which is specific for multi-label classification problems. To apply the optimization methods in different tasks, those must be replaced by other...
particular task-specific factors. On the other hand, the infrastructure-related data used to optimize the distribution process are not specific for the solved problem. Those are rather dependent on the underlying technology and the infrastructure used for the model building. In our case, we used the GridGain framework deployed on standard laboratory machines, which enabled us to directly obtain the needed data. Many other distributed computing frameworks have similar capabilities, and most of the data could be obtained directly from the OS deployed on the machines.

In order to utilize the optimization approach to a wider range of text-processing tasks and models, semantic technologies could be leveraged. For this purpose, such a semantic model could be developed, which would address the necessary concepts related to task assignment–infrastructure description, data description, and model description. The semantic model then could be used to choose the right task-assignment strategy for particular distributed model according to specific conditions related to the underlying distributed architecture and processed data.

The experiments were performed on selected standard datasets that are frequently used in the text classification domain. Our main objective was not to focus on the performance of the particular classifiers themselves, but rather to compare how the presented optimization methods could enhance the existing distributed classifiers with no task or environment optimization implemented. From this perspective, the conducted experiments could serve as a proof of concept that the application of optimization solutions to other distributed classification model implementations could bring similar benefits to their performance.

7. Conclusions

In this paper, we presented a comparative study of optimization methods used for task allocation and the improvement of distribution applied in the domain of multi-label text classification. Our main objective was to prove that the integration of the data characterizing the task complexity and computational resources can enhance the effectiveness of building distributed models and can optimize resource utilization. The developed and implemented methods were experimentally evaluated on standard text corpora, and their effectiveness was proved, especially when deployed on small-sized distributed infrastructures. The overall task completion time was significantly lower when compared with sequential solutions. It also performed well when compared with distributed model building with no optimization. The proposed approach is suitable for multi-class classifiers, as the task-related data are specific for that type of problem. To use the proposed methods in a wider range of text-mining tasks, a more general method of task-related data description has to be utilized. One of the possible approaches is to use semantic technologies, which could enable the construction of more generalized models applicable to tasks other than classification.

Author Contributions: Algorithm design, M.S. and M.O.; implementation, M.O.; experiments and validation, M.S. and M.O.; writing—original draft preparation, M.S.

Funding: This work was supported by Slovak Research and Development Agency under the contract No. APVV-16-0213 and by the VEGA project under grant No. 1/0493/16.

Conflicts of Interest: The authors declare no conflict of interest.

References


