UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE DISERTAȚIE

Unsupervised text feature selection using NSGA II with Hill Climbing local search

propusă de

Laura-Maria Cornei

Sesiunea: iunie, 2023

Coordonatori științifici

Lect. Dr. Croitoru Eugen, Conf. Dr. Breabăn Mihaela

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Unsupervised text feature selection using NSGA II with Hill Climbing local search

Laura-Maria Cornei

Sesiunea: iunie, 2023

Coordonatori științifici

Lect. Dr. Croitoru Eugen, Conf. Dr. Breabăn Mihaela

Table of Contents

Introduction

Text mining is a data mining sub-domain, with important applications in practice [1], focused on gaining useful information from textual data by using NLP techniques. It includes text feature selection and extraction, text summarization, text clustering and categorization.

The first step when analyzing textual data is the preprocessing phase, involving general operations such as stop words removal, lemmatization/stemming and other operations specific to the task. This step is often followed by a feature selection phase (in which a subset of relevant input features is selected) and/or feature extraction phase (in which a subset of new relevant features is created based on the given input features). A special case of the feature extraction methods is represented by the projection pursuit techniques [2], [3], aiming to extract low-dimensional projections of the data which reveal interesting properties. Criteria used in the project pursuit task include the Kurtosis [4] measure, the Holes Index [5], [6] and other metrics.

The feature selection/extraction phase is essential for reducing the high dimensionality of textual data by eliminating redundant, unnecessary and noisy features from the documents. This step increases the chances of performing a more meaningful future analysis of the textual data by avoiding the ”curse of dimensionality” and reduces the time and space complexity of subsequent text mining steps, such as classification and clustering.

The exponential increase of the textual unlabeled corpora (web documents, emails, etc.) has increased the need for improving the existing unsupervised feature selection and/or extraction techniques and discovering new ones.

Genetic Algorithms (GAs) [7] are meta-heuristics inspired by the process of biological evolution, with a potential of yielding high quality results for complex optimization and search problems. Genetic Algorithms are a promising alternative technique from the classical methods for solving the text feature selection problem, providing a global search for the optimum feature subset. However, using a single criterion (the GA's fitness function) for determining the best feature subset may not always yield good enough results.

Therefore, it may be useful to utilize multiple selection criteria [8], [9]. If the single used objective incorporates multiple feature selection metrics, there still remains the problem of weighting each employed metric. Multi-objective Evolutionary Algorithms, such as Non-dominated Sorting Genetic Algorithm (NSGA II [10]) and Strength Pareto Evolutionary Algorithm (SPEA II [11]), are suitable candidates for solving this issue. These techniques use the concept of Pareto dominance to organize the solutions in fronts, based on the degree to which they dominate and are dominated by other solutions.

In order to complement the global search conducted by the chosen (multi-objective) evolutionary strategy and potentially increase the overall performance of it, hybridizations with other meta-heuristics such as Sine Cosine Algorithm [12] and Grasshopper Optimization Algorithm [13] may be employed.

In this dissertation paper, we aim to address the unsupervised text feature selection task by making two substantial contributions. We have previously highlighted the challenges and benefits associated with this problem.

Firstly, we introduced a new unsupervised text feature selection method which sequentially applies NSGA II and Hill Climbing. The NSGA II meta-heuristic was used to provide an initial global search for the best feature subset(s), while the Hill Climbing strategy complemented it by performing a local improvement of the finally selected global optima. The chosen dominance criteria for the NSGA II algorithm were: an adapted version of the classical Mean Absolute Difference (MAD) relevance measure [14], [15] and the Holes projection pursuit index [5], [6] both weighted by the length of the current feature subset. The Hill Climbing method was utilized to reduce the number of redundant features in the chromosomes, redundancy of a term being measured as the sum of lexical and semantic similarities between the current term and the other features in the chromosome.

Experiments conducted on four real-world textual subdatasets indicated that our proposed approach achieved very good results in comparison to four state-of-the-art evolutionary unsupervised feature selection techniques [14], [15], [16], [17] and to one classical feature extraction method with proven effectiveness (Principal Component Analysis).

Secondly, we created a parallelized (distributed) version of the above-mentioned method in Spark [18], using operations over PySpark dataframes, due to their increased efficiency when compared to operations over RDDs. The preprocessing and solution evaluation steps were also reimplemented in Spark.

We tested our solution on the RaaS university's cluster. We realized various experiments on multiple benchmark textual datasets to asses the performance and the time efficiency of our approach. The obtained results confirmed that our proposed method reached performance levels that were comparable to or higher than those achieved by four recent and efficient evolutionary feature selection techniques and by the Principal Component Analysis feature extraction method.

The rest of the paper is organized as follows: section 1 discusses related work; section 2 presents the textual datasets used in the experiments and the preprocessing steps performed over them; section 3 introduces the proposed unsupervised feature selection method based on NSGA II and Hill Climbing; section 4 describes the parallelized version in Spark of the proposed approach, section 5 contains the experimental results. The last section includes final discussions, conclusions and ideas for future improvements.

Chapter 1

Related work

This chapter is divided into two comprehensive subsections. The first one describes multiple state-of-the-art and other representative evolutionary methods utilized for solving the text feature selection problem. The second subsection, on the other hand, covers various techniques utilized for parallelizing single and multi-objective genetic algorithms. At the end of each subsection, we present the contributions that we brought in comparison to the existent state-of-the-art techniques and other relevant methods.

1.1 Nature inspired methods for solving the text feature selection task

1.1.1 Review of nature inspired techniques for addressing the text feature selection problem

Evolutionary Algorithms for solving the feature selection problem have been studied intensively for years [19]. Most of the first proposed solutions utilized single objective Genetic Algorithms (GA), while subsequent algorithms made use of meta-heuristics such as versions of Particle Swarm Optimization (PSO) [14] and Ant Colony Optimization (ACO) [20] adapted to this problem, combined multiple meta-heuristics [13] or considered multiple objective functions [21].

In the case of supervised feature selection, the fitness function of the proposed single objective solutions usually combined the size of the subset of features with the accuracy/error rate provided by a classifier in a weighted sum [22], [23]. These criteria were also commonly utilized in multi-objective algorithms such as NSGA II [21]. Multiple other interesting objective functions have been taken into account over the last years, such as the entropy of sensitivity and specificity of a classifier [24] and the multivariate Relative Discriminative Criterion that minimizes the Pearson correlation between the features while maximizing the relative discriminative criterion [9].

Regarding the unsupervised feature selection, the options of choosing the objective function(s) are a lot more limited. The proposed Genetic Algorithms usually employed fitness functions that generalized the definitions of filter unsupervised metrics such as Term Variance [25] or TF-IDF [26] to adapt them to work with a subset of features. For example, a simple method of generalization consisted in performing a sum or a mean over the individual scores of the metric for each term to determine the score of the group of features [26]. An extension of this approach involved introducing a threshold and ignoring the scores calculated for a document and a group of features if the percentage of terms from the group that were also present in the document was lower than the given threshold [25].

Another very popular evolutionary strategy utilized for unsupervised feature selection was Particle Swarm Optimization, as mentioned in [19] Even though PSO was traditionally used for global numerical optimization problems, adaptations of it, such as Binary Particle Swarm Optimization made it possible for it to operate on a discrete space to solve the feature selection problem [27] To avoid the premature convergence problem of PSO [28], common solutions that included tuning the inertia weight and/or using an adaptive constriction factor were also utilized for the feature selection task [29]. Adaptations of PSO utilizing variable length particles [8] were proposed to eliminate the constraint of using fixed length representations and to improve the performance of PSO for this task.

Besides the mentioned meta-heuristics, recent solutions for the unsupervised text feature selection problem included Binary Grey Wolf Optimizer (BGWO) [15], Binary Multi-Verse Optimizer (BMVO) [16] and Krill Herd Algorithm (KH) [17].

The most commonly used fitness function for those three techniques (BGWO, BMVO, KH) and also for multiple versions of PSO was the mean absolute difference (MAD), used with the TF-IDF weighting scheme [14], [15], [16]. This optimization criterion was used under the assumption that the i_{th} candidate solution of PSO/ BGWO/ BMVO/ KH from the population encoded the selected features (words) from the i_{th} document; the length of each solution was equal to the total number of unique words existent in all documents. Important limitations of this popular approach include memory and run-time efficiency issues and the loss of generality of the solution (because a different subset of features is selected for each document).

1.1.2 Our contribution

Our work has multiple purposes: firstly, it eliminates the above-mentioned limitations, by proposing a different representation for the solutions and for the associated operators. Secondly, it broadens the choices of objective functions for this task, by bringing into attention two efficient and effective criteria: an adapted version of MAD and the Holes projection pursuit index. Thirdly, it complements the lack of literature regarding the usage of multi-objective meta-heuristics for the unsupervised feature selection problem by introducing a solution based on NSGA II. It also introduces a local improvement strategy using Hill Climbing, that aims to eliminate redundant features based on a proposed measure combining lexical and semantic similarities. Lastly, we provide a thorough comparison of our method with four state-of-the-art evolutionary techniques for solving the unsupervised feature selection problem and with one classical feature extraction method, our solution showing very good results.

1.2 Methods and strategies for parallelizing NSGA-II and GA evolutionary techniques

Our proposed technique sequentially applies NSGA II with Hill Climbing to solve the problem of unsupervised text feature selection. To implement a distributed version of this method, we first need to examine the ways in which we can parallelize the components of the proposed approach.

The Hill Climbing meta-heuristic itself cannot be parallelized, as it is a trajectory method that performs a local search by iteratively improving the current solution. However, the task of applying Hill Climbing separately on each of the best chromosomes previously determined by NSGA II is *embarrassingly parallel*.

The advantages offered by the evolutionary techniques (for e.g., single and multi-objective genetic algorithms) usually come at the cost of a large time complexity. In case of genetic algorithms, the process of evolving the population and evaluating it using the fitness function over a series of generations is often a time-consuming one. In case of multi-objective evolutionary algorithms such as NSGA II, the non-dominated sorting and crowding distance sorting steps further add to the complexity.

To improve the time complexity of these population based methods (GA and NSGA II), multiple methods for parallelizing them have been developed over the years [30], [31]. In this section, we discuss and compare classic models of parallelization commonly used for GAs and we review related work regarding strategies of parallel computing for the GA and NSGA II evolutionary methods.

1.2.1 Classic models of parallelization for Genetic Algorithms

As mentioned in the literature [32], [33], [31], there are currently three classical models for parallelizing a GA, that can also be extended to multi-objective evolutionary methods such as NSGA II and to other evolutionary approaches:

- *The Master Slave model* (also called *Global Parallelization model*). In this model, there is one master node that manages the population by applying GA operators (e.g., crossover, mutation) and by performing the selection; when it comes to fitness evaluation (usually the most expensive step in the algorithm), the master distributes the chromosomes to slave nodes that compute the fitness for the assigned individuals and receives the results based on which it performs the selection in order to create the new population.

- *The Island model* (also called *Coarse-Grained Parallelization model/ Multiple-deme model*). This model operates over subpopulations (of the initial population) called islands. More exactly, each slave node runs all evolutionary steps of a GA (crossover, mutation, fitness evaluation, selection) on a given subpopulation. In order to preserve the diversity and promote evolution, chromosomes can be migrated between subpopulations according to a set of rules.

- *The Cellular model* (also called *Fine-Grained Parallelization model/ Grid model*). As in the previous model, the GA is applied separately over subpopulations, however now a subpopulation represents an adjacent neighbourhood in a distributed grid where each node in the grid can retain one or more chromosomes.

The Cellular model requires massively parallel hardware usage in order to be computationally efficient. The communication overhead between the different nodes also represents a challenge. Due to these reasons, the first and second model are usually preferred.

The Master Slave model brings the advantage that it leads to solutions of the same quality as the sequential GA, however it is offers less parallelization than the Island model. In contrast, the higher degree of parallelization provided by the Island model comes at the cost of potentially reaching suboptimal solutions due to diversity issues. Another downside of the Island model might be the communication overhead resulted from the migration process of the chromosomes between the subpopulations.

Regardless of the utilized parallelization model, there is still the issue of accessing data for fitness evaluation and/or for applying genetic operators (such as mutation). If the data is retained in the memory of each node this increases the performance, however there might be memory issues in case of large quantities of data that need to be retained. If the data is shared in real time between nodes, then there might appear communication overhead issues. Cluster computing frameworks such as Hadoop [34] and Spark [35] are indicated to be used to deal more easily with such problems and other problems related to parallel computing (scalability, availability, etc.).

1.2.2 Review of parallel computing techniques for GA and NSGA II

The literature offers many examples of parallelized GAs for solving various optimization problems, however there are considerably fewer papers concerning the parallelization of multi-objective algorithms such as NSGA II. Due to reasons mentioned in the previous section, most proposed solutions adapt the Master-Slave or the Island model. Next, we will discuss several significant parallelization approaches and highlight the most remarkable contributions of the authors.

In [36], the authors proposed one synchronous and two asynchronous versions of the Master-Slave model for the NSGA II algorithm. In the classical synchronous version of the parallel NSGA II, the master node simply distributed a set of chromosomes to each worker node. In case of both asynchronous versions of NSGA II, the master node initially sent P individuals to the workers, where P was the number of available workers; as soon as a worker returned the evaluation for a chromosome, it was assigned a new individual, until all chromosomes were evaluated. The authors proved that this strategy made full use of the computational power of the slave nodes and increased the performance the algorithm. The difference between the two proposed asynchronous approaches for parallelizing NSGA II was that one of them was generational (meaning that from the current population a new separate population of offspring was created and afterwards both populations were combined and the selection was applied) and the other one was steady state (meaning that the generated offspring were directly inserted in the current population). The authors deployed their approach in Sparrow, a framework for distributed computing also created by them.

The authors of paper [37] indicated how GAs could be implemented using the Map Reduce paradigm (Hadoop framework) and demonstrated the scalability of their approach for large problem sizes. Each generation of a GA was associated to a Map Reduce job as follows: the map function received as input a key which was a unique identifier of the chromosome and a dummy value, determined the fitness function of the chromosome and emitted the chromosome as key and its fitness as value. A random partitioner was utilized to shuffle the chromosomes and distribute them randomly to reducers. This step was essential as utilizing the custom hash partitioner would have introduced an unnatural constraint affecting negatively the convergence of the genetic algorithm. In the reduce phase, the authors performed a tournament selections of size S (without replacement) and uniform crossovers: in case of each reducer, chromosomes were added into a buffer until the number of elements in the buffer reached the limit S; then the selection operator was applied to choose two chromosomes; finally a uniform crossover was applied over the two selected chromosomes and the reducer emitted two pairs (key, value), where the key was one of the created offspring and the value was a dummy value. At each iteration, the best chromosome from the population was written on Hadoop Distributed File System (HDFS).

Paper [38] presented a Master-Slave parallel NSGA-II algorithm for solving the multi-objective optimal power flow problem. The authors implemented their solution using an MPI (Message Passing Interface) approach. To share data necessary for the fitness computation, the Master node broadcasted the required information through MPI; the slave nodes saved the received data locally. A difference from the classical Master-Slave approach was that the master node also calculated the fitness for a subset of chromosomes from the population.

The authors of paper [30] described an implementation of a parallel GA in Spark which was utilized to solve the test data generation problem. Authors highlighted the main advantages of Spark over other frameworks or libraries for parallel computing such as MPI or PVM (Parallel Virtual Machine): easy setup, automatic fault-tolerance and the option to test the solution locally or on a cluster with minimal configurations. The proposed approach followed the Island model, such that in the map phase several GAs running over different random populations were instantiated and in the reduce phase the best chromosomes generated by each GA were collected.

In paper [39], it was proposed a Spark implementation of a GA for sensor placement in large scale drinking water distribution systems. In the Driver program, the population was initialized as an RDD and parallelized into different partitions. The map function was used to calculate the fitness of each chromosome. After the process of fitness evaluation was finalized, a selection followed by the application of the crossover and mutation operators was performed in the Driver code. The process was continued until the stopping condition was met.

Paper [32] presented a Spark based framework for parallelizing GAs using the Island model. More exactly, the population of chromosomes was distributed on separate partitions of an RDD and multiple GAs ran on each partition, such that the application of genetic operators did not take into account the individuals from the other partitions. Once every $MigrationInterval$ generations, best $MigrationRate$ chromosomes were broadcasted from each partition and each partition replaced its worst chromosome by the fittest broadcasted solution. Authors compared their solution to the sequential GA and proved that the gained time performance was significant and that the error rate didn't increase considerably.

In paper [33], the authors introduced two generic implementations of a GA on the Apache Hadoop framework and in Spark. In contrast to other discussed papers, the presented implementation of a generic GA in Hadoop included a new strategy for performing the crossover, more exactly several crossover pools were created based on the chromosomes with the same fitness values (the chromosomes with the same fitness were grouped together via a Map Only job which evaluated each individual and emitted its fitness as key and the chromosome as value). After the crossover pools were created, another Map Reduce job was ran: each mapper worked on a chromosome pool and generated multiple new chromosomes by applying the crossover and mutation operations; the reducers aggregated and evaluated the newly generated chromosomes. The main procedure performed the selection operation over the new offspring chromosomes collected from the reducers. The process was repeated iteratively until the stopping condition was met. The Spark implementation of the GA resembled very much the Hadoop implementation in terms of logic.

Paper [31] described a parallel bi-objective NSGA II algorithm implemented in Spark for solving the supervised feature selection problem. The two objectives taken into account were the AUC metric and the cardinality of the chosen subset of features. The proposed solution followed the Island model: first, the initial population was split into k randomly selected overlapping subpopulation of the same size and all hyperparameters were broadcasted to all islands; next, each subpopulation was evolved independently and in parallel by each worker for $mGen$ maximum generations; at the end of $mGen$ iterations, the chromosomes were migrated between the subpopulations and the process was repeated. In order to perform the migration, first, all chromosomes were collected from the subpopulations, then non dominated sorting and crowding distance sorting was performed and top chromosomes were selected to form the new population. This new population was again split into separate subpopulations that would be evolved in separate islands. The process continued until a termination condition was met. The Authors also described parallel versions of other multi-objective algorithms, such as MOEA/D (Multi-objective Evolutionary Algorithm Based on Decomposition) and NSPSO (Non-Dominated Sorting Particle Swarm Optimizer). On the DDoS dataset, the presented NSGA II parallelized approach outperformed the two other proposed solutions for the supervised feature selection problem.

1.2.3 Our contribution

Our proposed approach involved the sequential application of NSGA II and Hill Climbing. To parallelize the NSGA II evolutionary method, we used the *Master Slave* model, due to the advantages that it brought in comparison to the other two models (highest quality of the solutions, relatively fast model) [40]. As we have previously mentioned, the parallelization of the second part of the proposed method consisted in simultaneously executing the Hill Climbing meta-heuristic over a set of chromosomes selected as being the best solutions by NSGA II.

The implementation was realized in Spark due to the numerous advantages that this framework brings in contrast to other tools or libraries for parallel computing (speed, flexibility when it comes to choosing the running mode - local / pseudo-distributed / cluster, automatic fault-tolerance, etc.) [30], [41]. We chose to perform the operations over PySpark dataframes, due to their increased efficiency when compared to operations over RDDs.

We tested our solution and showed that it achieved close results to the ones obtained by the non-distributed version; the execution time was also significantly lower, as expected. We also performed experiments on a large dataset with 1000 documents (BBC 1000), showing that our solution obtained good results when compared to PCA. In the last performed experiment, we proved that our distributed feature selection approach had, in many cases, a positive impact on multiple classification and clustering algorithms.

Chapter 2

Used datasets and preprocessing steps

This section provides details regarding the textual datasets utilized to test our proposed solutions and the preprocessing steps performed over them. To evaluate both the non-distributed and the parallelized proposed approaches, we conducted separate experiments over datasets differing in size; the non-distributed version was tested on smaller datasets (with at most 350 documents), while the parallelized version was also evaluated on a considerably larger dataset (with 1000 documents).

2.1 Used datasets

To compare our non-distributed approach with multiple state-of-the-art feature selection and feature extraction techniques for solving the unsupervised text feature selection task, we utilized four real-world textual subdatasets:

1. $Reuters200$, a subset of 200 randomly chosen news documents from Reuters-21578 [1] - belonging to 4 categories: *alum, acq, barley, bop*.
2. $20Newsgroups300$, a subset of 300 randomly chosen news documents from 20Newsgroups [2] – belonging to 3 categories: *talk.politics.guns, sci.med, soc.religion.christian*.
3. $Reuters350$, a subset of 350 randomly chosen news documents from Reuters-21578 – belonging to 10 categories: *zinc, yen, silver, orange, cotton, coffee, coconut, housing, potato, gold*.
4. $BBC1000$, a subset of 1000 randomly chosen news documents from the BBC news corpus [3] - belonging to 5 categories: *business, entertainment, politics, sport,*

[1]https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection
[2]https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups
[3]https://www.kaggle.com/datasets/shivamkushwaha/bbc-full-text-document-classification

tech. This dataset is perfectly balanced, each category being associated to 200 instances.

We chose these datasets as they were representative for the task, frequently used in NLP research and already utilized in experiments by the authors of the state-of-the-art methods used for the comparison.

2.2 Non-distributed preprocessing steps

The non-distributed preprocessing was applied to corpora with at most 350 documents. The first step consisted in transforming the documents using the NLTK [4] and Spacy [5] Python libraries. We adjusted the Spacy tokenizer to not split compound words by hyphen. For each document, we converted all terms to lowercase, we removed one and two letter words, as well as special characters and tokens (such as URLs, math formulas, numbers which were not part of a word). Stop words were removed by using the stop words list provided by NLTK combined with a custom larger stop words list[6]. We then lemmatized the remaining words.

We further proceeded by performing several preprocessing steps to obtain fast future computations of the two used NSGA II objective functions. To effectively determine the Holes projection pursuit index, we retained for each term the list of document ids in which it appeared, creating an inverted index. In case of the adapted MAD objective function, we determined a MAD score for each unique term in the vocabulary, using a weighting based on an improved version of the TF-IDF score [42], as detailed below.

The TF-IDF metric considers the frequency of a term inside a document ($tf(t,d)$), but also an inverse document frequency ($idf(t)$), which has the purpose of making features appearing in many documents weight less, as they are supposed to refer to potentially more general irrelevant terms. The classic formula is given below:

$$tfIdf(t,d) = tf(t,d) * idf(t) \tag{2.1}$$

In case of datasets with categories of varying sizes, terms from classes with larger sizes are assigned lower IDF values. Moreover, rare and irrelevant terms may be assigned higher scores than relevant terms appearing more frequently. To solve these issues, authors of paper [42] proposed an improved version of TF-IDF. The IDF term

[4]https://www.nltk.org/
[5]https://spacy.io/
[6]https://www.ranks.nl/stopwords

was adapted to reduce the bias induced in case of corpora with unbalanced categories as follows:

$$idf(t) = \log\left(\frac{nb\_docs + 1}{a_{df}(t) + 1}\right) \tag{2.2}$$

$$a_{df}(t) = \frac{(df(t) - \overline{df})^2}{nb\_terms} \tag{2.3}$$

where $\overline{df}$ represents the mean of all $df(t)$ values for each term t, nb_terms denotes the vocabulary size and nb_docs the total number of documents.

The adapted MAD score for a term t, based on the improved TF-IDF weighting scheme, was calculated as follows:

$$MAD_{term}(t) = \frac{1}{nb\_docs} \sum_{i=1}^{nb\_docs} |tfIdf(t, d_i) - \overline{tfIdf(t)}| \tag{2.4}$$

$$\overline{tfIdf(t)} = \frac{1}{nb\_docs} \sum_{i=1}^{nb\_docs} tfIdf(t, d_i)| \tag{2.5}$$

where $tfIdf(t, d_i)$ represents the improved TF-IDF score for the term t and the i_{th} document d_i, and $\overline{tfIdf(t)}$ is the mean of the improved TF-IDF scores for the term t.

The proposed MAD score differed from the original formula [14], [15], [16], as it was adjusted to enable the discovery of the best feature subset(s) for the entire collection of documents, rather than determining individual solutions for each document. The weighting scheme was also an improved version of TF-IDF, not the classical one.

We finally min-max normalized the adapted MAD scores, bringing them in the range [0,1], such that these scores could also be used in a roulette wheel selection during the population initialization step.

2.3 Parallelized preprocessing steps

The parallelized preprocessing included the same initial steps as the ones from the non-distributed version: tokenization, keeping only tokens of length at least 2 which were formatted as words, lemmatization based on POS tags, stop words removal.

To implement these first preprocessing steps in Spark, a pipeline of transformations was applied over the Pyspark dataframe containing the documents. In cases in which a transformer was not already implemented for the preprocessing subtask, we defined our own transformers.

Next, we constructed an inverted index inv_idx, retaining for each term t the $MAD_{term}(t)$ score and the list of ids of the documents in which the term appeared. This information was useful for speeding up the computation of the MAD and Holes objective functions of the NSGA II algorithm.

To build the inverted index, we started by applying several aggregations over the dataframe of preprocessed documents, determining for each term t:

- its document frequency ($df(t)$)
- the list of frequencies of appearance of t in each document d_i ($tfIdf(t, d_i)$)
- the list of ids corresponding to documents in which t appeared

Then, we utilized an user defined function (UDF) to calculate for each term t its inverse document frequency ($idf(t)$), starting from its document frequency ($df(t)$). We finally used another UDF to compute the normalized adapted MAD score of t ($MAD_{term}(t)$) using the inverse document frequency ($idf(t)$), as well as the list of frequencies for each document d_i ($tfIdf(t, d_i)$). Because of its large size, we stored the obtained inverted index on HDFS (Hadoop Distributed File System).

Chapter 3

Proposed unsupervised text feature selection method

The proposed method employed two metaheuristics applied sequentially, namely NSGA II and Hill Climbing. NSGA II was utilized to conduct an initial search for the best feature subset(s) on a global scale. The Hill Climbing strategy was further used to enhance this process by performing a local improvement of the selected global solutions. The following subsections present details concerning the design of the two strategies.

3.1 Global search using NSGA II

The pseudo-code of the NSGA II based global search method for finding the optimal feature subset(s) is given below. The algorithm followed the classical NSGA II [10] steps, namely it started with an initial population that was iteratively improved until a stopping condition was met.

```
pop = initialize_population(pop_size, min_init_ch_len, max_init_ch_len,
    tournament_size)
while (!stop_condition(pop, max_nb_of_generations, $\sigma_{lim}$)):
    determine_fronts(pop)
    pop = selection_using_fronts_and_crowding_sorting(pop_size, pop)
    parents = iterated_tournament_selection(pop)
    pop = nondestructive_crossover(parents, pop)
    pop = nondestructive_mutation(pop)
return get_chromosomes_front1(pop)
```

Listing 3.1: Pseudocode of the nondistributed proposed algorithm

At each generation, every chromosome C was assigned a rank indicating the front, in the $determine\_fronts$ function, utilizing the principle of Pareto dominance. The rank was determined based on the list of dominated chromosomes, as well as the number of chromosomes that dominated C [10]. Lower ranks were associated to chromosomes that attained better values for the two utilized objective functions.

Next, the function $selection\_using\_fronts\_and\_crowding\_sorting$ was used to select pop_size chromosomes as part of the new population, by choosing chromosomes with the lowest ranks. In case only some chromosomes needed to be selected from a specific front, the split decision was made by taking into account the crowding distance [10]. Thus, from the chromosomes belonging to the same front, the ones that were located in a less crowded region were chosen. In case of the first iteration, the number of chromosomes in the population was already equal to pop_size, but in subsequent generations, new chromosomes were added in the population during the mutation and the crossover steps.

Finally, the parents for the crossover were selected in the $iterated\_tournament\_selection$ function and the crossover and mutation operators were applied non-destructively to obtain the new population.

After the stopping condition was met, the algorithm returned as output the chromosomes from front 1 from the final population, which indicated the best feature subsets selected according to the optimized objective functions.

The next subsections include comprehensive information regarding the mentioned steps.

3.2 Population initialization and stopping condition

The initial population consisted of pop_size chromosomes, each chromosome corresponding to a feature subset. The initial length of the chromosomes varied between $min\_init\_ch\_len$ and $max\_init\_ch\_len$ inclusively. A gene in a chromosome was represented by a vocabulary term. We considered this representation of the genotype to avoid the limitations discussed in section 2. Each gene was chosen using either a tournament selection of size $tournament\_size$ or a Roulette Wheel approach, both with equal probability. Regardless of the utilized method, the normalized MAD score of a term was used as a fitness function for the selection.

The algorithm was stopped if the maximal crowding distance stabilized over L generations or if a number of $max\_nb\_of\_generations$ iterations was exceeded. In paper [43], authors discussed and demonstrated empirically the superiority of choosing a stop criterion based on the stabilization of the maximal crowding distance and proposed the following formula for the termination condition, that we also utilized in our work:

$$\sqrt{\frac{1}{L}\sum_{i=1}^{L}(d_l - \overline{d_l})^2} < \sigma_{lim} \tag{3.1}$$

where d_l is the maximal crowding distance computed at generation l, $\overline{d_l}$ is the mean of the maximal crowding distance computed over L generations and σ_{lim} is a threshold chosen empirically.

3.3 NSGA II objective functions

The procedure for determining the fronts ($determine\_fronts$ function) took into account two objective functions: an adapted version of the Mean Absolute Difference (MAD) and the Holes projection pursuit index, both weighted by the length of the subset of features. For a chromosome ch, the criteria were calculated as follows:

$$MAD(ch) = \frac{1}{\sqrt{nb\_feats}} \sum_{i=1}^{nb\_feats} MAD_{term}(t_i) \tag{3.2}$$

$$Holes(ch) = \frac{1}{\sqrt{nb\_feats}}(1 - \frac{1}{nb\_docs} \sum_{k=1}^{nb\_docs} \frac{1}{e^{\frac{1}{2}x_k \cdot x_k}}) \tag{3.3}$$

where nb_feats represents the number of features in the chromosome ch and $MAD_{term}(t_i)$ is the adapted MAD score for the i_{th} feature of ch and nb_docs is the total number of documents. The $x_k \cdot x_k$ value retains the sum of squares of the Doc2Vec [44] embeddings for the terms from ch that also appear in the k_{th} document.

The proposed MAD score was modified from the original formula [14], [15], to allow the identification of the optimal feature subset(s) for the entire set of documents, instead of generating separate solutions for each document.

The Holes index [5] is a metric commonly utilized in projection pursuit [2] due to its efficient calculation and robustness. The maximization of this index enables one to discover clustering patterns in the data. To apply this index for a textual dataset in the feature selection scenario, the first step is to keep only the selected terms in each document and then to represent numerically these reduced documents as vectors, using a weighting scheme such as Doc2Vec. x_k represents the vector of embeddings corresponding to the k_{th} reduced document, while $x_k \cdot x_k$ is the dot product between x_k and itself (also equal to the magnitude of the vector squared).

We chose to integrate the length of the feature subset in the two objective functions instead of creating a third objective function that minimized the length of the chromosome, in order to avoid the convergence towards non-dominated solutions with a very large number of terms. Small-scale experiments showed that the square root function was a good option for weighting the total number of features in each chromosome.

3.4 Crossover and mutation operators

The $iterated\_tournament\_selection$ function repeated a binary tournament selection in order to choose pop_size chromosomes as parents for the crossover step. When comparing two chromosomes, the one with a lower rank, or with a higher crowding distance in case of equal ranks, was selected.

The selected parents were grouped two by two and an order preserving crossover was applied for each pair. The operation was non-destructive, so the resulted offspring were added to the population.

To perform the crossover for two chromosomes $ch1$ and $ch2$ a cutting point was chosen randomly for each of them, as well as a parameter s indicating a side (left or right). The first offspring was created by copying the values from the side s of chromosome $ch1$ and appending on the other side the values from $ch2$ that were not already added in the offspring. The second offspring was similarly created by reversing the roles of ch1 and $ch2$. This procedure ensured that only unique terms were retained in each of the resulting chromosomes.

We applied a single mutation on each chromosome obtained after the crossover, generating an offspring that was also added in the resulting population. There were three types of possible mutations: add mutation (a randomly chosen term in the vocabulary, not present in the current chromosome, was added at a random position), delete mutation (a term at a randomly chosen position was deleted) and modify mutation (a randomly selected gene was replaced with a randomly chosen vocabulary term not present in the chromosome). Each type of mutation had an equal probability of being chosen.

3.5 Local search using Hill Climbing. Final selected subset of features

The Hill Climbing method was applied over the chromosomes from front 1 selected by the NSGA II meta-heuristic in order to provide a local improvement of the currently chosen feature subsets, by discarding redundant features from them.

For each selected chromosome, $nb\_of\_iterations\_Hill\_Climbing$ steps were performed, at each step the current chromosome being compared to at most $nb\_neighbors$ neighbors and replaced with the first neighbor that was at least as good as it.

To obtain the neighbors of a chromosome, a list with the size $min(nb\_neighbours, chromosome\_length)$ of most redundant terms from the chromosome was determined. With an equal probability of 1/3, each neighbor was generated by deleting one redundant term from the chromosome, by replacing one redundant term from the chromosome with a random term from the vocabulary or by adding to it one random term from the vocabulary.

The more similar a term was in comparison to the other features in the chromosome, the more redundant it was considered to be. The assumption made was that related features in terms of lexical and semantic similarity appeared in the same contexts and didn't provide much additional information. The similarity score between two terms $t1$ and $t2$ was measured by summing a normalized version of the Levenshtein distance with two WordNet types of semantic similarity (WuPalmer and shortest path similarity) [45]:

$$sim(t1, t2) = norm\_lev(t1, t2) + wup(t1, t2) + path(t1, t2) \tag{3.4}$$

where

$$norm\_lev(t1, t2) = \frac{max(len(t1), len(t2)) - lev\_dist(t1, t2)}{max(len(t1), len(t2))} \tag{3.5}$$

In the formulas above, $lev\_dist(t1, t2)$ is the Levenshtein distance between $t1$ and $t2$, $norm\_lev(t1, t2)$ is a normalized in the [0,1] range version of it, $len(t)$ represents the number of letters in term t and wup and $path$ denote the WordNet similarity functions. The WuPalmer metric takes into account the depths in the WordNet taxonomy of the synsets corresponding to each term and of their lowest common hypernym to calculate the similarity, while the path similarity determines the shortest paths between the two synsets corresponding to each term.

The similarity score of a term t in the chromosome was calculated as the sum of all similarity scores between t and the other features in the chromosome. The terms with the highest similarity scores were considered to be the most redundant.

Lastly, we retained the new chromosomes resulted after applying the Hill Climbing method. To obtain the final set of selected features, we computed the frequency of the terms belonging to these chromosomes and finally chose at most $nb\_selected\_terms$ terms with the highest frequencies.

Chapter 4

Parallelized version in Spark of the proposed method

4.1 Cluster architecture and configurations

To implement the parallelized version of the proposed method in Spark, we worked on the university's RAAS (Research-As-A-Service) cluster [1]. The parallelized implementation followed closely all the steps presented in the non-distributed version.

The used stack configuration contained 4 worker nodes and 1 driver (master) node, where all the nodes had the m1.large flavour (8 VCPUs, 8GB of RAM, 500 GiB of disk memory). Each worker node had associated 2 executors, each executor being allocated 3 cores and 2GB of RAM. The Driver node was allocated 7 cores and 5GB of RAM. On each node the RAM and number of core resources were less than the maximum threshold (8 GB of RAM and 8 cores), as 1 core and 2-3 GB of RAM were already used or could be used by the OS, Yarn, Hadoop and Spark.

4.2 Pseudo-code of the distributed algorithm

Below is given the pseudocode for the distributed procedure combining the NSGA-II and Hill Climbing algorithms. We used a middle way between the master slave model (in which only the fitness function was computed in parallel) and the island model (in which multiple workers ran the algorithm in parallel).

More exactly, we parallelized the population initialization, evaluation, crossover, mutation and Hill Climbing steps, but performed the steps for determining the fronts and for the crowding distance sorting on the driver node. In this way, we could ex-

[1] http://raas-is.uaic.ro/Pagini/Servicii.aspx

ploit at maximum the parallelization benefits without risking to suffer a loss in the evaluation performance.

```
samples = get_samples(inv_idx, init_pop_sample_cnt,
                      init_pop_big_sample_size_percent)
pop_df = initialize_population(pop_size, min_init_ch_len, max_init_ch_len,
                               samples, tournament_size)
while !stop_condition(pop_df, max_nb_of_generations, L, $\sigma_{lim}$):
    pop_objf_df = evaluate_ch(pop_df, inv_idx)
    pop_objf_rank_df = determine_fronts(pop_objf_df.collect())
    selected_ch_df = crowding_sorting_and_selection(pop_objf_rank_df)
    ch_front1_df = get_ch_first_front(selected_ch_df)
    pop_df = crossover_and_mutation_spark(selected_ch_df)
ch_front1_HC_df = Hill_Climbing_local_search(ch_front1_df)
return get_features(ch_front1_HC_df)
```

Listing 4.1: Pseudocode of the distributed algorithm

4.3 Broadcasted data

As mentioned in the section 2.3, we used the determined inverted index inv_idx to achieve a faster computation of the two objective functions (Holes and MAD) used by the NSGA II algorithm. We broadcasted the inverted index, such that each node of the cluster had a copy of the shared data. In this way, we avoided the inefficiency and communication overhead of distributing this information along with each task. This solution sped up the execution of the distributed algorithm, as the objective functions calculation step was utilized extremely frequently.

4.4 Population initialization and stopping condition

To parallelize the non-distributed population initialization step, we started by creating $init\_pop\_sample\_cnt$ samples of the inverted index inv_idx, each sample having a size of $init\_pop\_big\_sample\_size\_percent$ percent out of the total number of elements in inv_idx. This step corresponds to the $get\_samples$ method from the pseudocode presented in section 4.2.

Next, we generated the initial population by applying a Spark udf (user defined function) on a dummy Spark dataframe of size pop_size, using the $withColumn$ transformation. This udf received as additional parameters: the minimum and maximum initial length of a chromosome ($min\_init\_ch\_len$ and $max\_init\_ch\_len$), a random sample of the inverted index (from the ones generated) and the tournament size $tournament\_size$, useful in the tournament selection step.

The purpose of this udf was to create a chromosome corresponding to a feature subset, with a random size between $min\_init\_ch\_len$ and $max\_init\_ch\_len$. Just as in the case of the non-distributed version of the algorithm, each gene in a chromosome was chosen based on its normalized MAD score using with equal probability either a tournament selection, either a Roulette Wheel selection. Finally, we removed possible duplicated terms from each chromosome.

The stopping condition remained unchanged from the condition presented in the non-parallelized algorithm.

4.5 Population evaluation. Determining the fronts

At each step of the NSGA II algorithm, the current population was retained in a single-column Spark dataframe pop_df, each row containing a chromosome. To evaluate the population, we used a udf that received as additional parameter the broadcasted inverted index and computed the Holes and MAD scores for a given chromosome. The utilized objective functions were identical to the ones used in the non-distributed version. This evaluation phase corresponds to the $evaluate\_ch$ method in the pseudocode from section 4.2.

We didn't parallelize the step of determining the fronts as it wouldn't have been feasible (all chromosomes needed to be compared with each other). Therefore, we performed this phase on the master node, by applying a $collect$ action over the dataframe pop_objf_df, containing the chromosomes' objective functions and their ids. After assigning ranks to each chromosome in a non-distributed manner, we recreated the distributed Pyspark dataframe containing also the ranks besides the original chromosome information.

4.6 Crowding sorting and selection

For this step, we utilized the PySpark dataframe $pop\_objf\_rank\_df$ containing the chromosomes, their associated objective functions values and ranks, constructed in the previous phase (determining the fronts).

We iteratively selected chromosomes with increasing ranks, as long as the number of all the selected chromosomes didn't exceed pop_size, using the $filter$ transformation. To retain the chosen chromosomes, we created a new dataframe $selected\_ch\_df$

that was repetitively extended with subsets of chromosomes using *union*.

In case we couldn't select all chromosomes with a specific rank, a crowding sorting was performed and only the chromosomes with the maximum crowding distance were chosen and added to the final dataframe. This specific step was executed on the master node (function *crowd_sorting* in the given pseudocode).

```
def crowding_sorting_and_selection(pop_objf_rank_df):
    selected_ch_df = empty_df()
    size_selected_ch_df = 0
    for front_nb in [1 .. max_front_nb]:
         ch_with_front_df = get_ch_with_front(pop_objf_rank_df ,front_nb)
      if size_selected_ch_df + size_ch_with_front_df <= pop_size
          selected_ch_df = selected_ch_df.union(ch_with_front_df)
      else
          selected_ch_split_f_df = crowd_sorting(ch_with_front_df)
          selected_ch_df = selected_ch_df.union(selected_ch_split_f_df)
          break
     return selected_ch_df
```

Listing 4.2: Pseudocode for the crowding sorting and selection step

4.7 Crossover and mutation operators

We performed the crossover and mutation as operations over a Spark dataframe containing the chromosomes from the population.

To implement the crossover, two samples of size half of the total population were extracted and joined. To join them, we first associated indices to both samples using the $ZipWithIndex$ transformation and then we utilized *join*. We finally obtained a dataframe with two columns, $ch1$ and $ch2$, each of them retaining the selected chromosomes for the crossover. The operation was applied over the $ch1$ and $ch2$ columns using an udf, creating a third column *crossover*, which contained pairs of resulted child chromosomes.

These pairs of chromosomes were split on separate records using *explode* and a mutation was applied over each chromosome using another udf, creating a *mutation* column in the Spark dataframe.

The chromosomes from the initial population were united (*union*) with the chromosomes from the *crossover* and *mutation* columns, resulting in a Spark dataframe containing the final population.

The mutation and crossover procedures were identical to the ones presented in the non-parallelized version of the algorithm.

4.8 Local search using Hill Climbing. Final selected subset of features

The distributed local search strategy involved applying the non-distributed Hill Climbing chromosome improvement procedure in parallel, over each chromosome from the front 1. To implement this, the non-parallelized approach was utilized as a udf over the PySpark dataframe ch_front1_df, containing the chromosomes with rank 1. A new column ch_front1_HC, created after applying the udf, included the resulted chromosomes obtained after the local search.

Just as in the non-parallelized version, the final subset of selected features was obtained by choosing the most frequent terms from the chromosomes of the ch_front1_HC column. This phase was associated to the $get\_features$ method in the pseudocode from section 4.2.

Chapter 5

Experiments for the classic and parallelized proposed methods

In this section we discuss the results obtained when comparing the two versions of our proposed solution (classical and parallelized) with multiple state-of-the-art and other relevant unsupervised feature selection and feature extraction techniques.

5.0.1 Experiments for the classic (non-distributed) proposed method

This section outlines the outcomes of the experiments conducted on three textual subdatasets (Reuters200, 20newsgroup300, Reuters350) to assess the effectiveness of our suggested approach and to compare it with a general effective unsupervised feature extraction method (PCA using Doc2Vec embeddings) and with 4 state-of-the-art evolutionary unsupervised feature selection techniques [14], [15], [16], [17].

The experiments involved clustering the current dataset using two versions of the K-Means algorithm: K-Means++ and Spherical K-Means [46], the last technique being a popular alternative for clustering textual data due to the usage of the cosine similarity measure [14], [16]. To analyze the clustering performance, we compared the original clustering assignments with the determined ones and further computed the mean and standard deviation for multiple evaluation metrics (Accuracy, Precision, Recall, Adjusted Rand Index (ARI), F1-measure).

In order to conduct the tests for our solution, we kept only the selected features in each document, vectorized the documents using Doc2Vec, applied the two versions of the clustering algorithms and computed the mentioned metrics.

In case of PCA, we first applied the Doc2Vec model over the entire collection of documents, then determined the PCA components using the Doc2Vec scaled vectors and finally used the clustering algorithms over the identified components to calculate the evaluation criteria.

For all tests, the Doc2Vec model worked with vectors of size 300 and a window size of 5, while the number of executed epochs was 20; the clustering methods were initialized 20 times in order to not let the initialization of the centroids influence significantly the results and the maximum number of iterations was 500.

To obtain statistically significant results, we ran both PCA and our method 30 times for each dataset.

Taking into account that the sizes of the used datasets were comparative, we used the same hyper-parameters for our propose approach, for all three textual collections. The utilized hyperparameters, given in Table 5.1, were empirically determined by running small scale experiments.

Table 5.1: Hyperparameters used for the proposed method

Parameter	Value
pop_size	100
$min\_init\_ch\_len$	10
$max\_init\_ch\_len$	50
$tournament\_size$	10
$max\_nb\_of\_generations$	250
L	40
σ_{lim}	0.02
$nb\_of\_iterations\_Hill\_Climbing$	10
$nb\_neighbours$	30

A characteristic of our method was that the number of finally selected features, corresponding to the $nb\_terms\_selected$ parameter (and also to the $Nb.of features$ column), could be varied according to the desired reduction rate. The other evolutionary techniques used for the comparison provided a fixed number of features for a given dataset; they also utilized only a single version of the K-Means algorithm (either classical or spherical) for all the experiments.

To compare our approach with the four state-of-the-art techniques, we used similar testing conditions: the same version of K-Means and a number of selected features very close to the number provided by those techniques. When performing comparisons with PCA, we varied both the number of selected features and the clustering method.

Table 5.2 indicates the comparative results for our proposed approach, the PCA method and FSPSOTC (Feature Selection technique based on PSO for improving Text Clustering) [14] on the $Reuters200$ dataset. When using the K-Means clustering, our solution obtained comparable or clearly better results than PCA; when using the Spherical K-Means clustering, our technique substantially outperformed PCA on all metrics. Our approach also achieved considerably superior results compared to FSPSOTC, attaining an increase of more than 25% for the Accuracy score, more than 8% for the Precision metric and around 23% for the Recall score.

Table 5.3 summarizes the results on the $20Newsgroups300$ dataset for the presented method, PCA and two state of the art evolutionary feature selection techniques (Binary Gray Wolf Optimizer [15] and Binary Multi-Verse Optimizer [16]). Our approach obtained a significantly higher F1-Measure score in comparison to the BGWO and the BMVO methods (0.51 vs. 0.34 and 0.36). When compared to PCA, it achieved similar results for the K-Means clustering and noticeably superior results on the Accuracy, Precision and F1-Measure scores for the Spherical K-Means clustering; the Recall scores for the Spherical clustering version were lower, however, due to the high standard deviation of the PCA method, further statistical tests were required to draw a clear conclusion.

Table 5.4 contains the outcomes of the experiment conducted on the Reuters350 dataset using the Spherical K-Means clustering. In this final experiment, we compared our method with PCA and with a state-of-the-art technique called MHKHA (Membrane-driven Hybrid Krill Herd Algorithm) [17]. The results showed that our approach attained comparative or better results than PCA. When compared to MHKHA, our method achieved substantially better results on the Accuracy metric (an improvement of almost 35%), comparative results on the Recall measure and a lower score on the Precision and F1-measure metrics (a decrease of almost 6%, and 3% respectively).

Table 5.2: Comparison between the proposed method, PCA and FSPSOTC on Reuters200 dataset

Method	Clustering algorithm	Nb. of features	Accuracy	Precision	Recall	ARI
Proposed method	K-Means	300	0.6424 (0.0299)	0.3625 (0.0295)	0.5598 (0.0260)	0.1951 (0.0416)
		800	0.6167 (0.0184)	0.3327 (0.0136)	0.5604 (0.0271)	0.1512 (0.0210)
		1000	0.6128 (0.0244)	0.3359 (0.0179)	0.5601 (0.0288)	0.1561 (0.0272)
	Spherical K-Means	300	0.8371 (0.0245)	0.6562 (0.0566)	0.7512 (0.0759)	0.5868 (0.0543)
		800	0.8431 (0.0158)	0.6594 (0.0421)	0.7853 (0.0777)	0.6071 (0.0360)
		1000	0.8399 (0.0209)	0.6547 (0.0515)	0.7792 (0.0767)	0.5995 (0.0473)
PCA	K-Means	300	0.5878 (0.0098)	0.3158 (0.0056)	0.5583 (0.0104)	0.1242 (0.0089)
		800	0.5914 (0.0044)	0.3190 (0.0036)	0.5623 (0.0022)	0.1302 (0.0063)
		1000	0.5919 (0.0104)	0.3177 (0.0056)	0.5531 (0.0151)	0.1267 (0.0085)
	Spherical K-Means	300	0.6773 (0.0929)	0.4240 (0.1155)	0.5617 (0.0305)	0.2577 (0.1432)
		800	0.6857 (0.0951)	0.4376 (0.1199)	0.5726 (0.0116)	0.2759 (0.1472)
		1000	0.6797 (0.0915)	0.4259 (0.1156)	0.5587 (0.0315)	0.2595 (0.1429)
FSPSOTC	Spherical K-Means	790	0.5845	0.5754	0.5518	-

Table 5.3: Comparison between the proposed method, PCA, BGWO and BMVO on 20NewsGroup300 dataset

Method	Clustering algorithm	Nb. of features	Accuracy	Precision	Recall	F1-measure
Proposed method	K-Means	1000	0.3537 (0.0035)	0.3464 (0.0008)	0.9858 (0.0040)	0.5124 (0.0002)
		1200	0.3532 (0.0034)	0.3462 (0.0007)	0.9850 (0.0042)	0.5124 (0.0002)
	Spherical K-Means	1000	0.6427 (0.0398)	0.4889 (0.0474)	0.5890 (0.0569)	0.5317 (0.0344)
		1200	0.6500 (0.0402)	0.4979 (0.0487)	0.6019 (0.0376)	0.5436 (0.0360)
PCA	K-Means	1000	0.3575 (0)	0.3472 (0)	0.9797 (0)	0.5127 (0)
		1200	0.3572 (0.0013)	0.3472 (0.0003)	0.9800 (0.0016)	0.5127 (9e-5)
	Spherical K-Means	1000	0.4589 (0.1023)	0.3682 (0.0212)	0.7308 (0.2512)	0.4716 (0.0419)
		1200	0.4588 (0.1025)	0.3681 (0.0212)	0.7305 (0.2520)	0.4714 (0.0421)
BGWO	K-Means	1065	-	-	-	0.3418
BMVO	K-Means	˜1150	-	-	-	0.3614

Table 5.4: Comparison between the proposed method, PCA and MHKHA on Reuters350 dataset using Spherical K-Means

Method	Nb. of features	Accuracy	Precision	Recall	F1-measure	ARI
Proposed method	1200	0.8904 (0.0063)	0.4676 (0.0272)	0.5505 (0.0379)	0.5015 (0.0340)	0.4405 (0.0387)
	1500	0.8907 (0.0070)	0.4675 (0.0282)	0.5534 (0.0392)	0.5062 (0.0285)	0.4454 (0.0318)
	1800	0.8897 (0.0106)	0.4649 (0.0407)	0.5464 (0.0447)	0.5013 (0.0361)	0.4400 (0.0411)
PCA	1200	0.8598 (0.0308)	0.3767 (0.0888)	0.4922 (0.0372)	0.4223 (0.0651)	0.3458 (0.0802)
	1500	0.8607 (0.0321)	0.3812 (0.0946)	0.4986 (0.0388)	0.4278 (0.0712)	0.3517 (0.0872)
	1800	0.8604 (0.0323)	0.3811 (0.0949)	0.4987 (0.0365)	0.4275 (0.0702)	0.3513 (0.0863)
MHKHA	1470	0.5441	0.5309	0.5449	0.5360	-

For all the three previously mentioned tables (5.2, 5.3 and 5.4), we also determined the associated plots representing the 95% confidence intervals for the true population mean of each utilized metric (Accuracy, Precision, Recall, etc.). This enabled us to achieve a more detailed comparison between our proposed method and PCA.

To determine the confidence intervals, we used the T-Distribution (and not the normal distribution), as the variance of the population was unknown [47]. The formula for calculating the confidence interval is given below:

$$CI = [\overline{X} + t * \frac{S}{\sqrt{n-1}}, \overline{X} + t * \frac{S}{\sqrt{n-1}}] \tag{5.1}$$

where $\overline{X}$ and S are the mean and the standard deviation of the sample of size n (in our case $n = 30$) and t is the t-value corresponding to $n - 1$ degrees of freedom and to a specific confidence level. For these experiments, we set the confidence level to 95%, thus using a t-value equal to 2.045.

Figures 5.1 to 5.5 visually describe the confidence intervals for each metric for all the three subdatasets. The confidence intervals for the proposed method (NSGA II hybridized with Hill Climbing) were depicted in green, while the ones for the PCA technique were represented in orange.

As it can be seen, when using the Spherical K-Means clustering, we can state with a level of confidence of 95% that the results obtained by the proposed strategy considerably outperformed the ones obtained by PCA (with a single exception for the Recall metric for the 20Newsgroup dataset).

When using the K-Means clustering, we could be 95% confident that in case of the Reuters 200 datasets the results for the accuracy, precision and ARI were undeniably higher for our proposed technique. The same thing happened when analyzing the confidence intervals for the recall metric on the 20Newsgroup300 dataset .

To conclude, the presented NSGA II hybridized with the Hill Climbing approach achieved a very good overall performance on the three used textual subdatasets. Compared to the PCA feature extraction method, it obtained considerably higher scores for almost all the utilized metrics when using the Spherical K-Means clustering and comparable scores when using the classical version of the K-Means. Our proposed solution also significantly outperformed the FSPSOTC (Feature Selection technique based on PSO for improving Text Clustering) [14], BGWO (Binary Gray Wolf Optimizer) [15] and BMVO (Binary Multi Verse Optimizer) [16] state-of-the-art evolutionary unsupervised feature selection techniques for all the used metrics and attained greatly better results than MHKHA (Membrane-driven Hybrid Krill Herd Algorithm)[17] for the accuracy metric and comparable results for the recall and F1-measure scores.

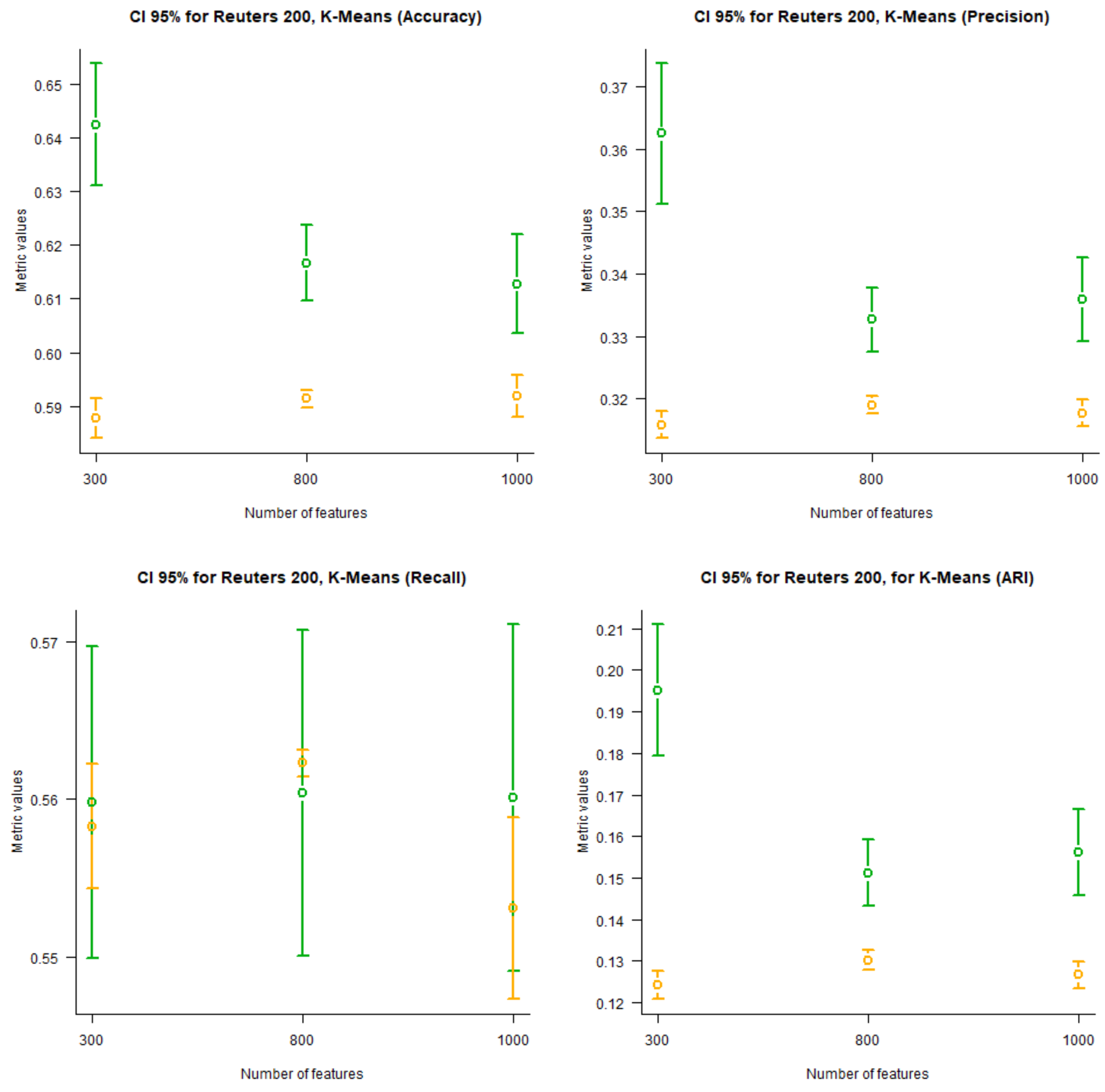

Figure 5.1: Confidence intervals (95%) for all the used metrics, comparing the proposed method (green) with PCA (orange), when using the K-Means clustering and the Reuters200 dataset

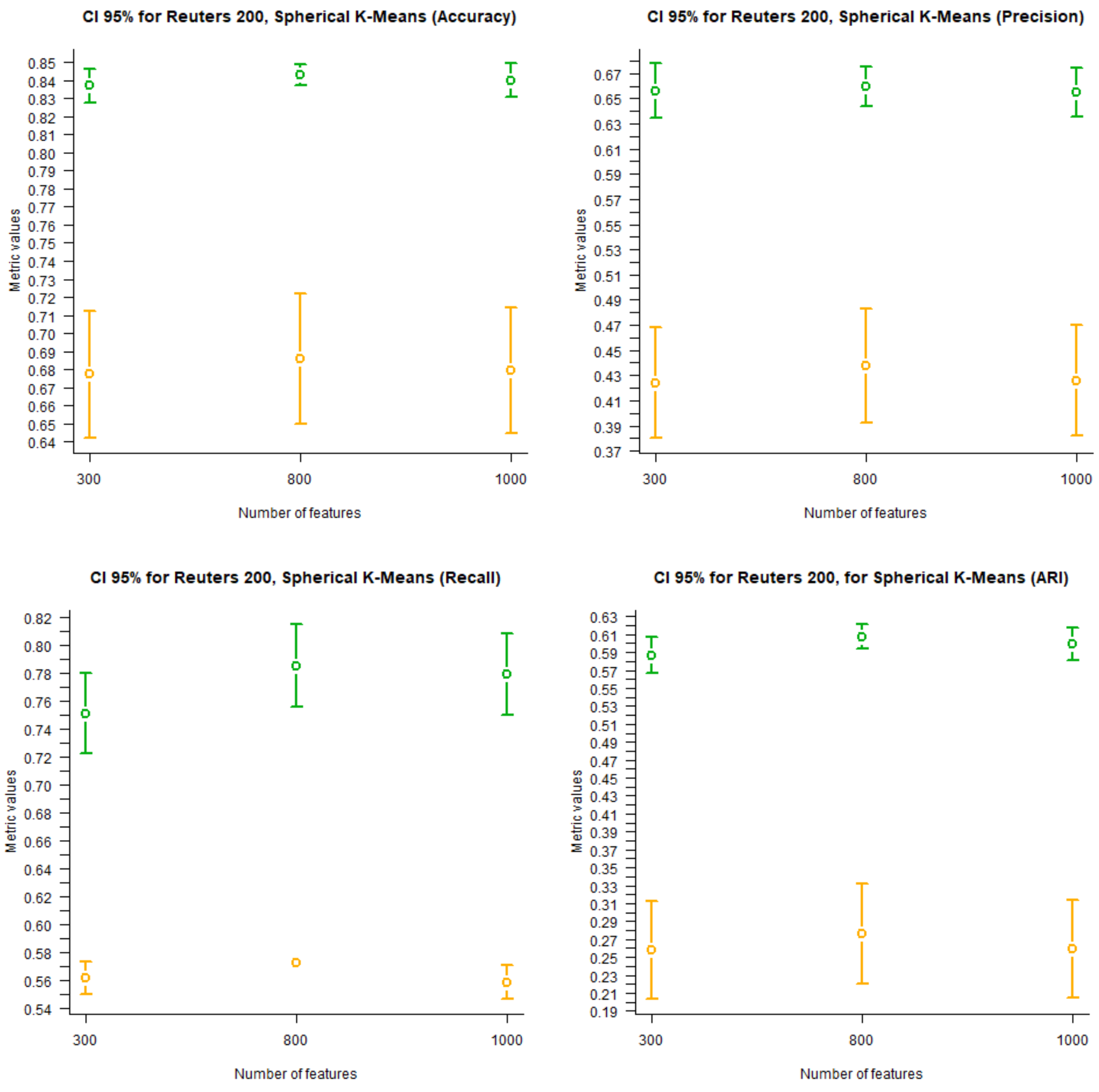

Figure 5.2: Confidence intervals (95%) for all the used metrics, comparing the proposed method (green) with PCA (orange), when using the Spherical K-Means clustering and the Reuters200 dataset

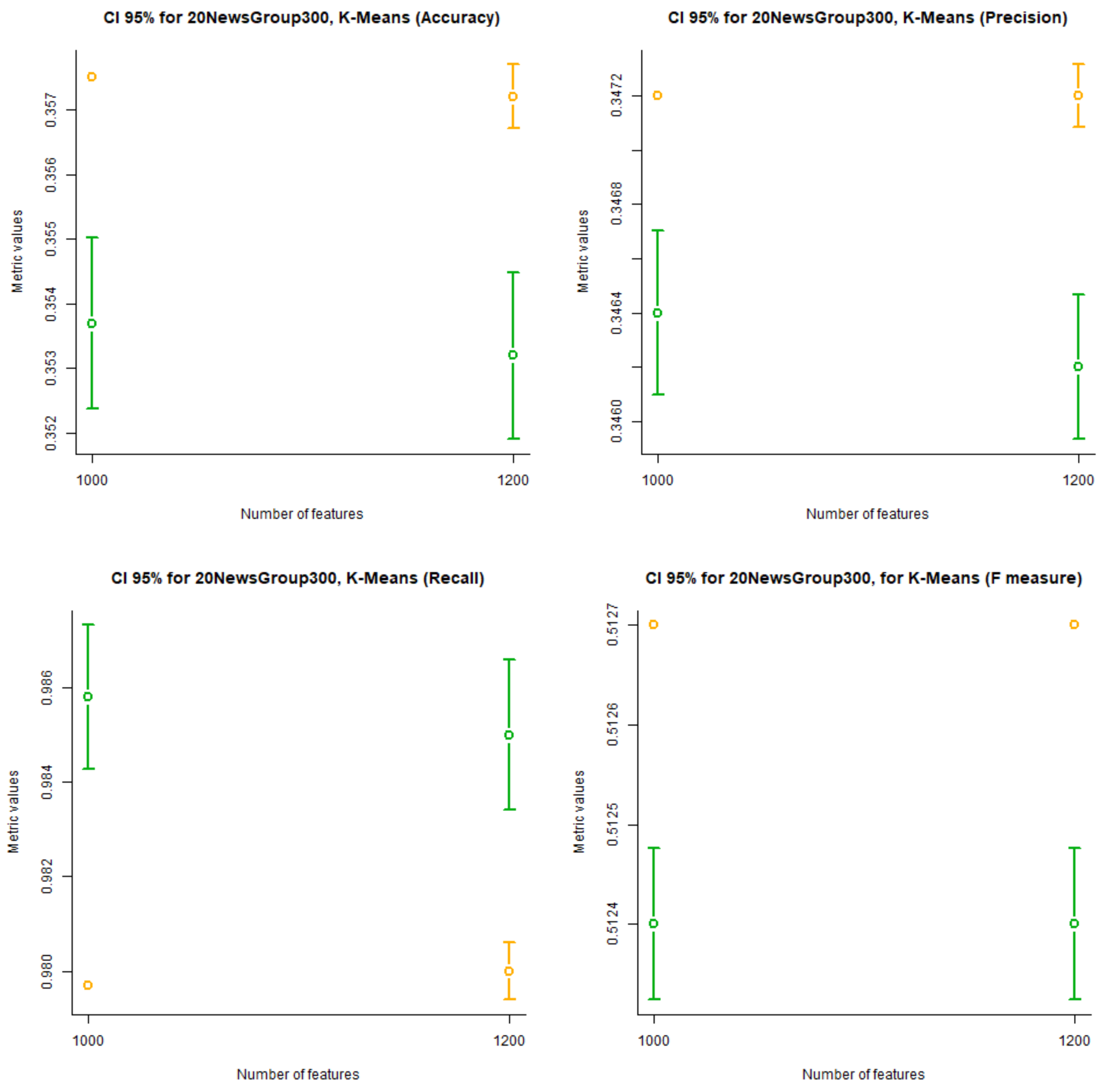

Figure 5.3: Confidence intervals (95%) for all the used metrics, comparing the proposed method (green) with PCA (orange), when using the K-Means clustering and the 20Newsgroup300 dataset

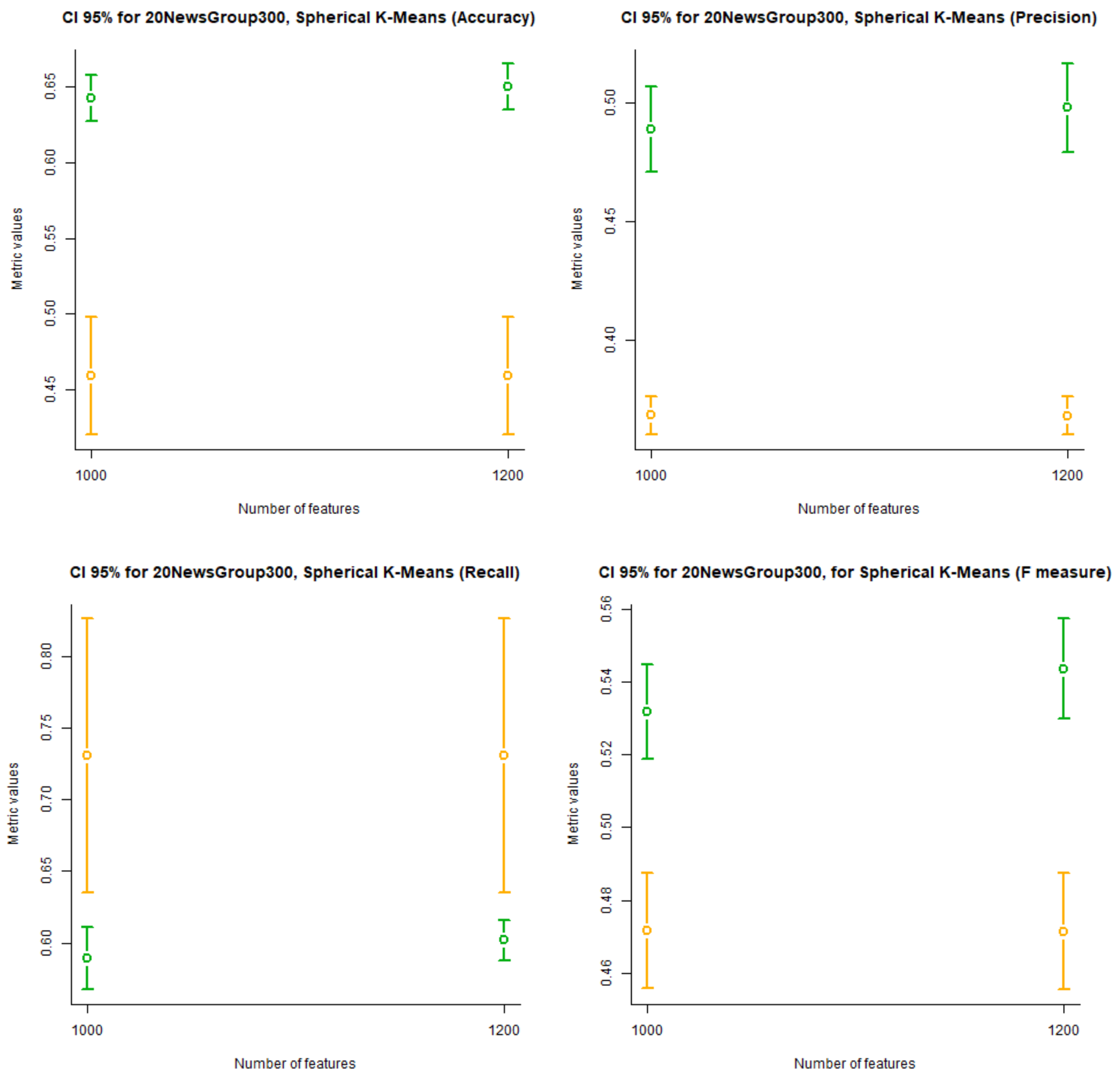

Figure 5.4: Confidence intervals (95%) for all the used metrics, comparing the proposed method (green) with PCA (orange), when using the Spherical K-Means clustering and the 20Newsgroup300 dataset

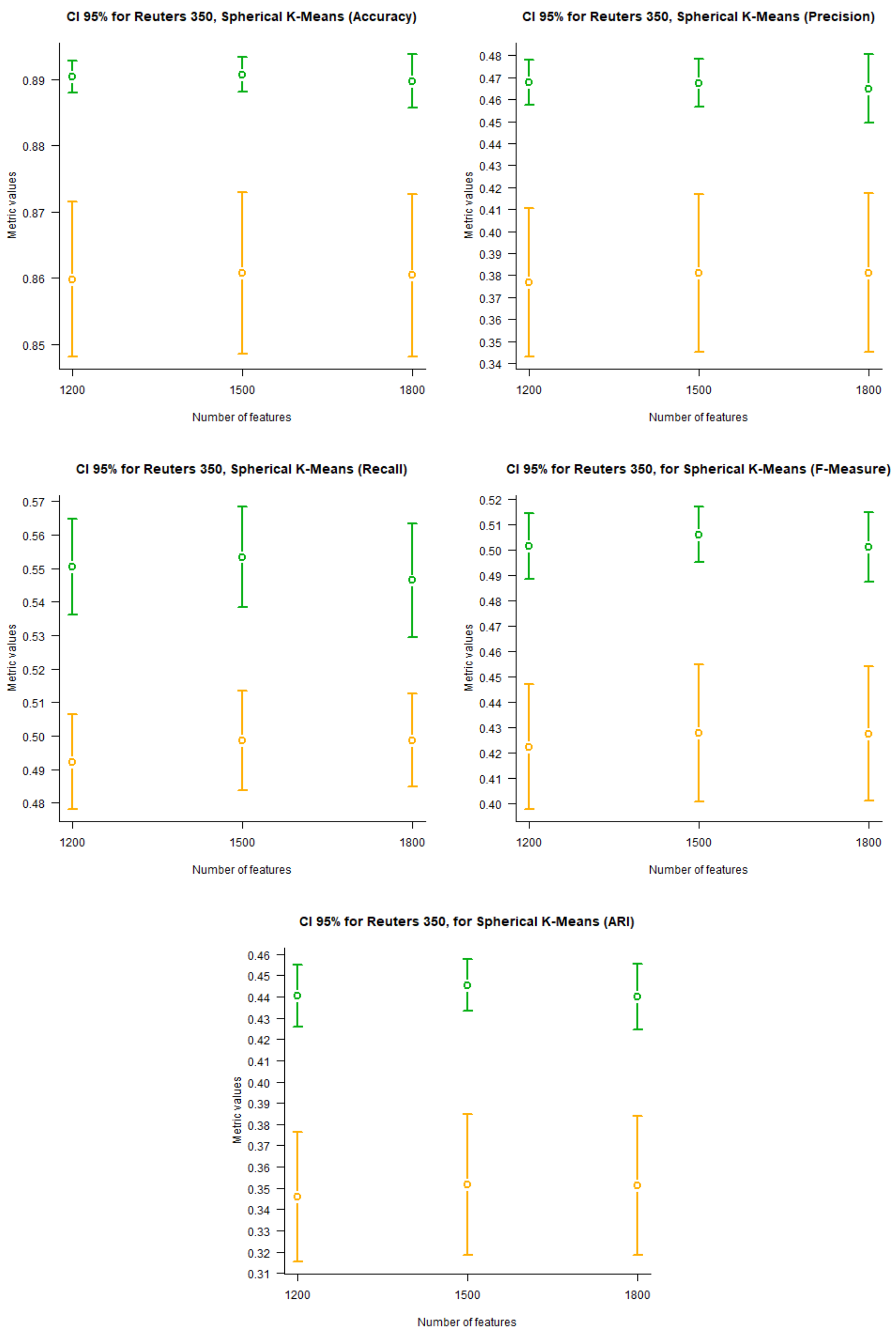

Figure 5.5: Confidence intervals (95%) for all the used metrics, comparing the proposed method (green) with PCA (orange), when using the Spherical K-Means clustering and the Reuters350 dataset

5.0.2 Experiments using the distributed proposed method

This section includes several detailed experiments, each one of them having a different purpose: to test the effectiveness and the correctness of our distributed approach, to evaluate the proposed parallelized solution on a larger dataset (with 1000 documents) and to evaluate the impact of the proposed method on clustering and classification.

Experiments to check the effectiveness and the correctness of the distributed implementation

Although we didn't measure the exact execution times of our proposed non-distributed approach, we estimated that between one and two days were needed to complete one run on one dataset of small size (Reuters200, Reuters350, 20Newsgroup300). In comparison to this estimation, the execution time for one run for the distributed solution was considerably shorter: less than one hour for each of the three datasets.

We also wanted to test the parallelized solution on a larger dataset with 1000 documents (BBC1000) and discovered that the execution time was also good (approximately three hours).

To obtain statistically significant results for the run times, we performed 30 iterations for each utilized dataset. The results from table 5.5 show that the distributed approach was significantly more efficient than the non-parallelized one and that it was also scalable.

Table 5.5: Execution times of the distributed approach for four datasets

Dataset	Execution time mean (Execution time stdev)
Reuters200	1874.8557 sec. (265.9606 sec.)
Reuters350	1780.9039 sec. (201.9898 sec.)
20NewsGroup300	2624.1588 sec. (102.9520 sec.)
BBC1000	11866.78420 sec. (357.5364 sec.)

To verify the correctness of the distributed implementation, we compared the metrics' values outputted by the parallelized technique with those previously obtained from the non-parallelized approach. We chose to perform this test on the Reuters350 dataset, as it was the one with the largest number of associated classes (10). When conducting the experiment, we fixed the number of features to 1500, as the results from the non-distributed approach showed very small variations (less than 1%) when changing the number of features to 1200 or 1800). We used the same hyperparameters and the same testing conditions when performing the experiment, as the ones presented in subsection 5.0.1.

Table 5.6: Comparison between the proposed distributed and non-distributed methods on the Reuters350 dataset using Spherical K-Means for 1500 features

Method	Accuracy	Precision	Recall	F1-Measure	ARI
Non-distributed proposed method	0.8907 (0.0070)	0.4675 (0.0282)	0.5534 (0.0392)	0.5062 (0.0285)	0.4454 (0.0318)
Distributed proposed method	0.8931 (0.0130)	0.4765 (0.0569)	0.5295 (0.0583)	0.5013 (0.0562)	0.4416 (0.0633)

The table 5.6 highlights the fact that the proposed distributed approach obtained close results to the ones from the non-distributed technique. This confirmed the expectations that the distributed solution followed closely the steps performed in the non-parallelized version.

Experiment evaluating the performance of the distributed version on a large dataset BBC1000

To perform the tests on the BBC1000 dataset, we followed the same steps described in the subsection 5.0.1.

More exactly, in case of our proposed distributed approach we retained only the chosen features from each document, transformed the documents into vectors using Doc2Vec, applied the two versions of the K-Means clustering algorithm, and calculated the specified metrics. In case of PCA, we utilized the Doc2Vec model on the complete set of documents, then we derived the PCA components by using the scaled vectors from Doc2Vec and lastly we applied the clustering algorithms on the identified components to compute the evaluation criteria.

The parameters for the K-Means clustering and for the Doc2Vec model were similar to those presented in subsection 5.0.1. To ensure that our results were statistically significant, we made 30 iterations for PCA and also for our method.

Table 5.7 contains the utilized hyperparameters for the proposed distributed technique. Because the number of documents in the BBC1000 dataset was significantly higher compared to the previous corpora, we increased the population size, the maximum number of generations and the number of iterations for Hill Climbing.

Table 5.7: Hyperparameters used for the proposed distributed method when using the BBC1000 dataset

Parameter	Value
pop_size	120
$min\_init\_ch\_len$	10
$max\_init\_ch\_len$	50
$init\_pop\_sample\_cnt$	10
$init\_pop\_big\_sample\_size\_percent$	0.4
$tournament\_size$	15
$max\_nb\_of\_generations$	400
L	40
σ_{lim}	0.02
$nb\_of\_iterations\_Hill\_Climbing$	15
$nb\_neighbours$	30

As it can be seen from the table 5.8, when experimenting on the BBC1000 dataset, the proposed distributed approach was outperformed by PCA when using the K-Means clustering, but achieved slightly better results when employing the Spherical K-Means clustering.

To check whether the results of our method were certainly better when using Spherical K-Means, we also performed t-tests for the mean of all the used metrics (accuracy, precision, recall, f1-measure and ARI), the results being summarized in Figures 5.6 and 5.7.

With a confidence level of 90%, we could state that in the case in which 5000 features were selected, the distributed technique obtained clearly better results for all the five utilized metrics.

When 3000 or 4000 features were selected or when a level of confidence of 95% was used, the confidence intervals of the two techniques were partially overlapping. This meant that no clear conclusion could be drawn, although the scores' means corresponding to our proposed approach were always considerably higher in value.

In conclusion, our proposed distributed technique outperformed PCA only in certain situations (for example, when using the Spherical K-Means clustering algorithm to select 5000 features). Apart from that, the results of our method, obtained for both of the clustering algorithms, were comparative or worse, but still high in value ($\sim$89% accuracy, $\sim$72% precision, $\sim$73% recall, $\sim$73% F1-measure and $\sim$66% ARI).

Table 5.8: Results of the proposed distributed method on the BBC 1000 dataset

Method	Clustering algorithm	Nb. of features	Accuracy	Precision	Recall	F1-measure	ARI
Proposed distributed method	K-Means	3000	0.8872 (0.0183)	0.7115 (0.0466)	0.7309 (0.0408)	0.7210 (0.0437)	0.6504 (0.0553)
		4000	0.8886 (0.0182)	0.7152 (0.0462)	0.7337 (0.0414)	0.7243 (0.0438)	0.6545 (0.0552)
		5000	0.8876 (0.0179)	0.7125 (0.0455)	0.7320 (0.0405)	0.7221 (0.0429)	0.6517 (0.0542)
	Spherical K-Means	3000	0.8919 (0.0229)	0.7261 (0.0586)	0.7371 (0.0480)	0.7315 (0.0533)	0.6638 (0.0678)
		4000	0.8850 (0.0288)	0.7082 (0.0735)	0.7236 (0.0618)	0.7156 (0.0672)	0.6436 (0.0853)
		5000	0.8924 (0.0261)	0.7274 (0.0658)	0.7378 (0.0568)	0.7325 (0.0614)	0.6651 (0.0779)
PCA	K-Means	3000	0.9433 (0.0024)	0.8555 (0.0061)	0.8607 (0.0056)	0.8581 (0.0059)	0.8227 (0.0073)
		4000	0.9427 (0.0028)	0.8538 (0.0076)	0.8596 (0.0064)	0.8567 (0.0070)	0.8209 (0.0087)
		5000	0.9428 (0.0025)	0.8540 (0.0065)	0.8598 (0.0058)	0.8569 (0.0061)	0.8211 (0.0076)
	Spherical K-Means	3000	0.8767 (0.0510)	0.6871 (0.1299)	0.7094 (0.1147)	0.6978 (0.1222)	0.6204 (0.1543)
		4000	0.8758 (0.0452)	0.6848 (0.1151)	0.7050 (0.1031)	0.6945 (0.1090)	0.6167 (0.1374)
		5000	0.8667 (0.0427)	0.6612 (0.1092)	0.6896 (0.0932)	0.6748 (0.1009)	0.5911 (0.1279)

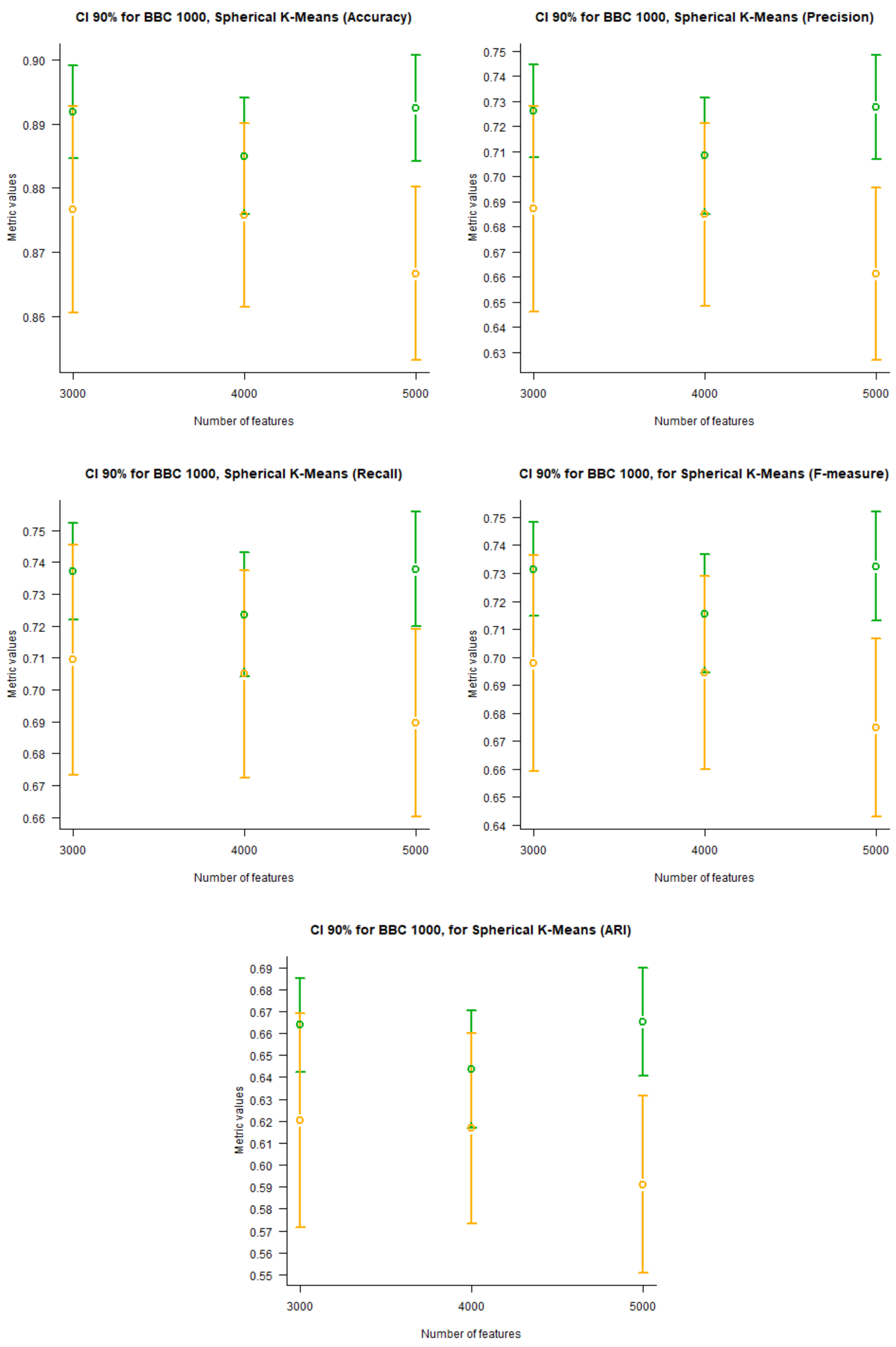

Figure 5.6: Confidence intervals (90%) for all the used metrics, comparing the proposed distributed method (green) with PCA (orange), when using the Spherical K-Means clustering and the BBC 1000 dataset

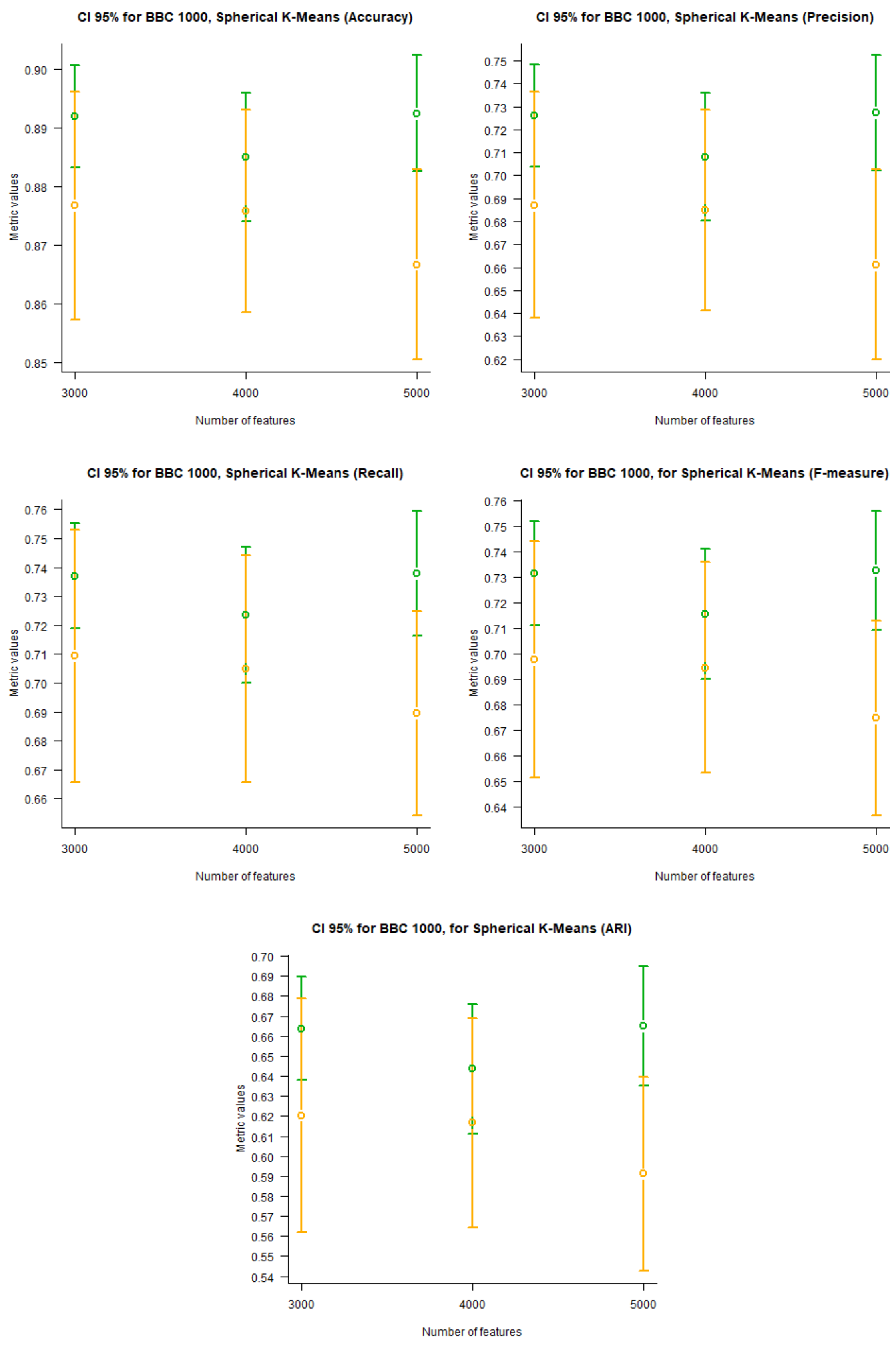

Figure 5.7: Confidence intervals (95%) for all the used metrics, comparing the proposed distributed method (green) with PCA (orange), when using the Spherical K-Means clustering and the BBC 1000 dataset

Experiment evaluating the impact of the proposed distributed method on clustering and classification

To analyze whether our proposed distributed feature selection technique could boost the classification and/or the clustering performance, we conducted the following experiment: we measured the performance of multiple classifiers and clustering algorithms in two different scenarios: one in which we applied the proposed distributed feature selection technique beforehand, and another one in which no feature selection was performed prior to classification/clustering. For implementation, we used the scikit-learn library [1].

For the experiment, we utilized three classification algorithms (Support Vector Machine, Logistic Regression, K-Nearest-Neighbors) and three clustering methods (EM, Hierarchical clustering with Ward similarity and Hierarchical clustering with Average Linkage similarity). To ensure the accuracy of the experiment, we made sure that we didn't select algorithms that performed an implicit feature selection step (for example, decision tree based strategies like Random Forest and XGBoost).

For each considered dataset, we vectorized the documents using classical TF-IDF embeddings to eliminate any potential bias that could have been introduced by more complex models, like Doc2Vec. To realize the evaluation we employed a repeated stratified k-fold cross-validation, with 4 splits and 10 repeats. Thus, 3 quarters of each dataset was used for training and one quarter for testing and each fold preserved the percentage of instances from each class.

Because the sizes of the used datasets were small, we couldn't make an automatic hyperparameter tuning, as this would have most probably led to overfitting. Therefore, we manually tuned the hyperparameters, aiming to avoid overfitting and underfitting scenarios.

Table 5.9 lists the values of the hyperparameters for the used classification and clustering algorithms in case of the Reuters350 and 20Newsgroup300 datasets. We included in the table only the hyperparameters for which we modified the default values offered by the scikit-learn library.

[1]https://scikit-learn.org/stable/

Table 5.9: Classification and clustering hyperparameters used for Reuters350 and 20Newsgroup300 datasets

Method	Sklearn hyperparameter	Hyperparameter value (Reuters350)	Hyperparameter value (20NewsGroup300)
SVM	C	0.5	0.5
Log. Regression	solver	saga	saga
	max_iter	300	300
	penalty	elasticnet	l1
	ratio	0.5	-
KNN	n_neighbors	15	30
EM	n_init	10	10
	covariance_type	full (default)	diag

In case of the Support Vector Machine (SVM) algorithm, we changed the regularization parameter C from the default 1.0 to 0.5 to make the model more general, allowing more points to be missclassified and thus avoiding overfitting.

When using the Logistic Regression classifier, for the Reuters350 dataset we chose the elastic net penalty, combining the strengths of both L1 and L2 regularizations. This was especially useful as in our case, the number of features (unique terms) was a lot greater than the number of samples (documents). In this way we could avoid extreme situations (the model taking into account too few or too many features). The $saga$ solver was the only one supporting the elastic net penalty. We fixed the number of maximum iterations to 300, instead of 100, to ensure that the algorithm converged.

When performing experiments for the 20newsgroup300 dataset, we kept the same parameters for the Logistic Regression model, except for the fact that we only used l1 regularization in order to take into account the large increase in the number of features (from 3125 for Reuters350 to 10753 for 20newsgroup300).

We fixed the number of neighbors in the K-Nearest-Neighbors algorithm to 15 for Reuters350, a value close to the square root of the total number of instances. For 20newsgroup300, this value was set to 30, as the number of class labels corresponding to this dataset was considerably higher than the number of class labels for the previous dataset (10 vs 3).

In case of the EM algorithm, we chose to make 10 initializations of the algorithm to obtain more stable results. Due to memory issues, we changed the default parameter *full* for the covariance matrix type to *diag* when using the 20newsgroup300 dataset.

Table 5.10 describes the impact of the distributed approach on classification and clustering when using the Reuters350 dataset. The number of features selected by the proposed parallelized technique was 600.

Our method brought a significant improvement in the scores of the three clustering methods and mild improvements in the scores of the used classification algorithms, compared to the case in which no feature selection was made.

Table 5.10: Results highlighting the impact of the distributed proposed method on classification and clustering for the Reuters350 dataset

Feature selection	Method type	Method	Accuracy	Precision	Recall	F1 score	ARI
Feature selection using the proposed distributed method	Classif.	SVM	0.8778 (0.0135)	0.9056 (0.0103)	0.8683 (0.0142)	0.8739 (0.0142)	0.7492 (0.0249)
		Logistic Regression	0.8960 (0.0177)	0.9092 (0.0181)	0.8888 (0.0183)	0.8904 (0.0192)	0.7910 (0.0266)
		KNN	0.8004 (0.0176)	0.8224 (0.0155)	0.7887 (0.0254)	0.7819 (0.0304)	0.6363 (0.0329)
	Clust.	EM	0.9184 (0.0108)	0.5953 (0.05351)	0.6181 (0.0403)	0.6062 (0.0463)	0.5608 (0.0524)
		Hierarchical (Ward)	0.8792 (0.0218)	0.4391 (0.0621)	0.6072 (0.0466)	0.5075 (0.0511)	0.4414 (0.0610)
		Hierarchical (Average Linkage)	0.8732 (0.0264)	0.4305 (0.0669)	0.6506 (0.0439)	0.5141 (0.0488)	0.4459 (0.0604)
No feature selection	Classif.	SVM	0.7938 (0.0054)	0.8796 (0.0047)	0.7653 (0.0061)	0.7785 (0.0062)	0.5775 (0.0088)
		Logistic Regression	0.8887 (0.0029)	0.9221 (0.0022)	0.8735 (0.0033)	0.8812 (0.0037)	0.7711 (0.0052)
		KNN	0.7788 (0.0025)	0.8199 (0.0005)	0.7678 (0.0043)	0.7666 (0.0040)	0.6054 (0.0024)
	Clust.	EM	0.8979 (0.0167)	0.5001 (0.0690)	0.5834 (0.0630)	0.5380 (0.0647)	0.4810 (0.0737)
		Hierarchical (Ward)	0.7691 (0.0)	0.2601 (0.0)	0.6928 (0.0)	0.3782 (0.0)	0.2707 (0.0)
		Hierarchical (Average Linkage)	0.8512 (0.0)	0.3672 (0.0)	0.6487 (0.0)	0.4690 (0.0)	0.3901 (0.0)

To check that there was indeed a mild improvement brought by our feature selection method when performing classification in case of the Reuters350 dataset, we realized a statistical testing experiment, the results being contained in figure 5.8. As it can be observed from the plots, our proposed distributed technique achieved significantly higher scores on almost all metrics. The only exception was the precision metric, for which our method obtained a lower value in case of the logistic regression classifier and a comparative value in case of KNN.

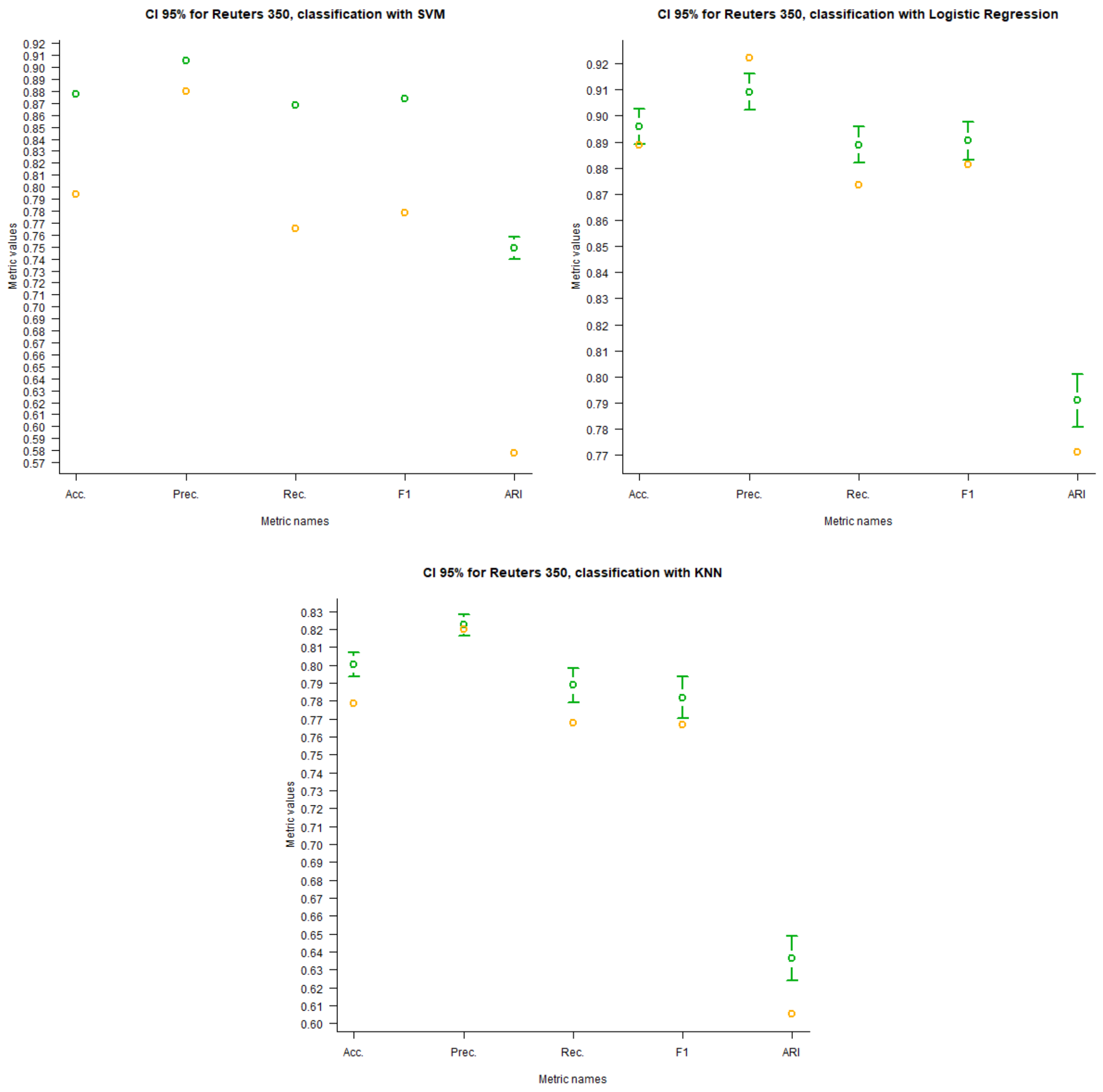

Figure 5.8: Confidence intervals (95%) for all the 5 used metrics, comparing the impact of using our feature selection method (green) with using no feature selection (orange), when using various classification algorithms on the Reuters 350 dataset

Table 5.11 highlights the impact of the distributed approach on classification and clustering when using the 20newsgroup300 dataset. The number of features selected by the proposed distributed approach was 800.

Table 5.11: Results highlighting the impact of the distributed proposed method on classification and clustering for the 20newsgroup300 dataset

Feature selection	Method type	Method	Accuracy	Precision	Recall	F1 score	ARI
Feature selection using the proposed distributed method	Classif.	SVM	0.7742 (0.0721)	0.8126 (0.0626)	0.7757 (0.0718)	0.7759 (0.0719)	0.4418 (0.1355)
		Logistic Regression	0.7604 (0.0507)	0.8071 (0.0499)	0.7625 (0.0508)	0.7628 (0.0508)	0.4069 (0.0894)
		KNN	0.7288 (0.0737)	0.7456 (0.0690)	0.7278 (0.0735)	0.7281 (0.0730)	0.3676 (0.1198)
	Clust.	EM	0.6040 (0.0565)	0.4238 (0.0616)	0.5090 (0.0746)	0.4580 (0.0524)	0.1479 (0.0993)
		Hierarchical (Ward)	0.4904 (0.0505)	0.3611 (0.0238)	0.6848 (0.0847)	0.4703 (0.0253)	0.0658 (0.0487)
		Hierarchical (Average Linkage)	0.3409 (0.0097)	0.3314 (0.0004)	0.9727 (0.0262)	0.4943 (0.0032)	0.0003 (0.001)
No feature selection	Classif.	SVM	0.6546 (0.0050)	0.8334 (0.0015)	0.6478 (0.0051)	0.6321 (0.0060)	0.2492 (0.0091)
		Logistic Regression	0.7378 (0.0024)	0.8405 (0.0015)	0.7423 (0.0024)	0.7403 (0.0024)	0.3517 (0.0062)
		KNN	0.8845 (0.0020)	0.8974 (0.0021)	0.8831 (0.0020)	0.8832 (0.0021)	0.6807 (0.0047)
	Clust.	EM	0.6001 (0.0268)	0.4286 (0.0532)	0.4553 (0.0569)	0.4413 (0.0539)	0.1074 (0.0814)
		Hierarchical (Ward)	0.3682 (0.0)	0.9105 (0.0)	0.3338 (0.0)	0.4885 (0.0)	0.0070 (0.0)
		Hierarchical (Average Linkage)	0.3402 (0.0)	0.3313 (0.0)	0.9735 (0.0)	0.4944 (0.0)	1.745e-05 (0.0)

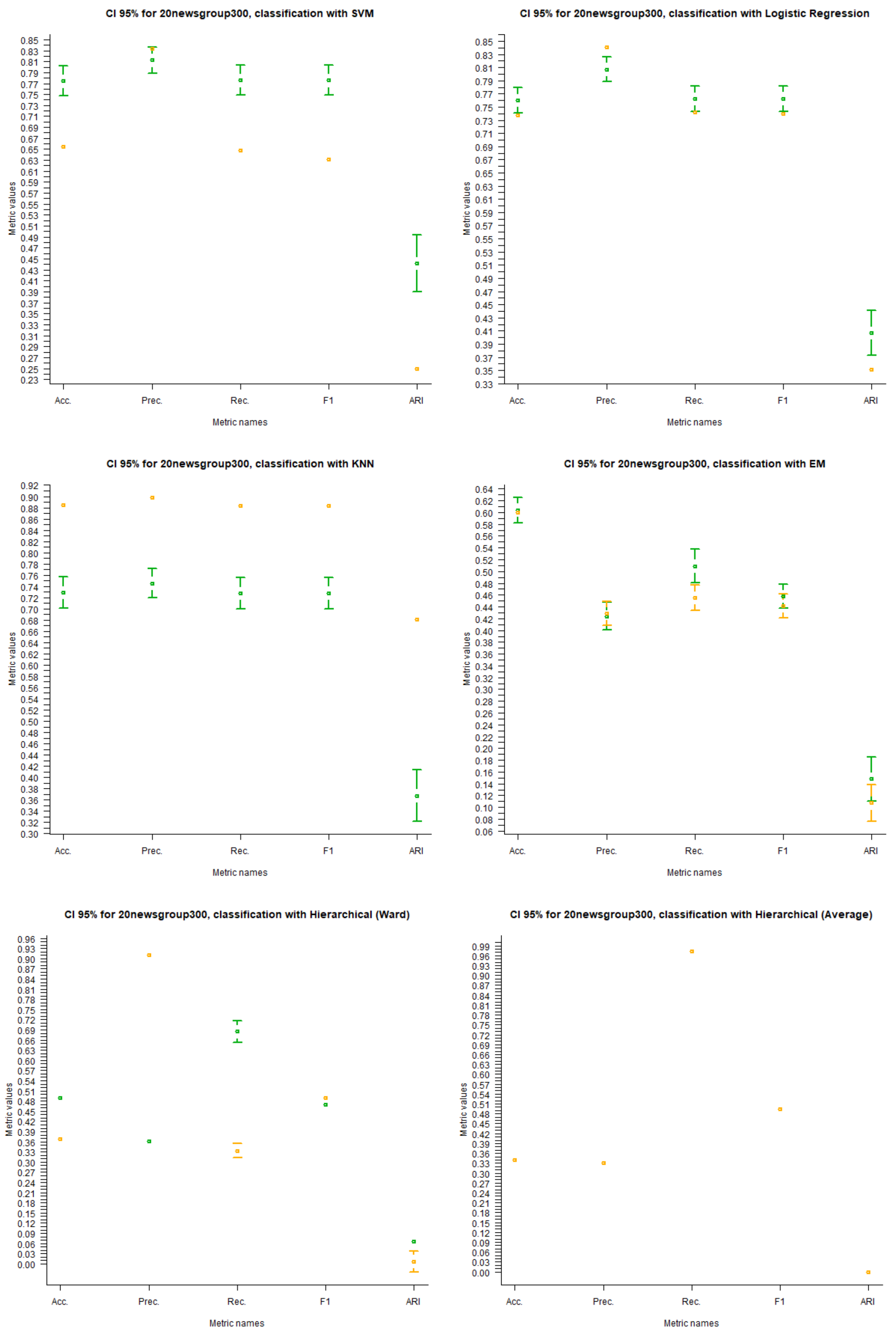

Figure 5.9: Confidence intervals (95%) for all the 5 used metrics, comparing the impact of using our feature selection method (green) with using no feature selection (orange), when using various classification and clustering algorithms on the 20newsgroup300 dataset

As it can be observed from the table 5.11 and also from Figure 5.9, which contains the associated statistical tests, our proposed feature selection technique considerably increased the performance of the SVM model for all the used metrics, when compared to the situation in which no feature selection was applied.

For Logistic Regression, EM and Hierarchical clustering using the Ward metric, we can affirm with a level of confidence of 95% that improvements were brought by our technique only for certain metrics (such as Accuracy, F1 and ARI in case of the Logistic Regression model, Recall in case of EM and Accuracy, Recall and ARI for Ward Hierarchical clustering).

For the Average Linkage Hierarchical clustering, our method brought no statistical significant improvements. Lastly, in case of the KNN algorithm, applying no feature selection yielded better results than applying the proposed feature selection method.

Conclusions for the experiments using the proposed distributed feature selection method

The first experiments performed in this section proved that our solution was correct, efficient and scalable.

The second experiment in which we compared PCA with our method on the BBC1000 dataset showed that our approach was comparable to PCA and was also able to considerably outperform this feature extraction technique under certain conditions (for example, when using the Spherical K-Means clustering algorithm to select 5000 features).

Taking into account the clustering and classification experiments conducted over both the Reuters350 and the 20newsgroup300 datasets, we can conclude that our proposed distributed features selection method brought, in many cases, a positive impact when applied before various classification and clustering models (SVM, Logistic Regression, KNN, EM, Hierarchical clustering with the Average and Ward linkages).

Conclusions and future improvements

This dissertation paper introduces a novel unsupervised text feature selection technique that uses the NSGA II multi objective algorithm for a global search (by taking into account as objective functions an adapted version of the Mean Absolute Difference relevance score and the Holes Projection Pursuit Index) and Hill Climbing for a local search (aimed at reducing the redundancy of the groups of features, by computing the lexical and semantic similarities between the terms in the same subset).

Experiments conducted on multiple textual subdatasets prove that our proposed solution outperforms three recent and efficient evolutionary feature selection techniques (FSPSOTC, BGWO, BMVO) and reaches performance levels that are higher than or comparable to those achieved by the Principal Component Analysis feature extraction method and by another evolutionary approach for feature selection (MHKHA).

We also created a distributed version of the proposed method in Spark and tested it in various ways. The first type of experiments confirmed that our solution obtains similar results to the non-distributed one and is efficient and scalable. Another experiment in which we compared our distributed method with PCA on the BBC1000 dataset demonstrated that our approach yields comparable results, while surpassing PCA significantly in specific circumstances. We finally performed an experiment to evaluate the impact of our approach on classification and clustering, reaching to the conclusion that our proposed feature selection method has, in many cases, a positive impact on the metrics' scores of various classification and clustering models.

Regarding future directions, we intend to generalize our approach by proposing an efficient evolutionary based solution for the unsupervised feature extraction task.

Another idea of improvement is to hybridize the current NSGA II algorithm with other metaheuristics. Simulated Annealing seems to be a viable option for local search, as it could enable our method to easily escape local optima in comparison to Hill Climbing, introducing however another hyperparameter search for choosing the cooling policy. We could also integrate NSGA II with population-based methods such as Grasshopper Optimization Algorithm [10] to improve the performance even further.

Acknowledgement

Data processing and analysis was realized with the support of project SMIS124759-RaaS-IS (Research as a Service Iasi), financed using the Operational Competitivity Programme. Many thanks to Mr. Cluci Marius-Iulian from the Faculty of Economics and Business Administration ("Alexandru Ioan Cuza" University) for helping me solve several issues related to the project configuration and resource management.

Bibliography

[1] S. Dang and P. H. Ahmad, "Text mining: Techniques and its application," *International Journal of Engineering & Technology Innovations*, vol. 1, no. 4, pp. 22–25, 2014.

[2] J. H. Friedman and J. W. Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Transactions on computers*, vol. 100, no. 9, pp. 881–890, 1974.

[3] C.-h. Chen, W. Härdle, A. Unwin, D. Cook, A. Buja, E.-K. Lee, and H. Wickham, "Grand tours, projection pursuit guided tours, and manual controls," *Handbook of data visualization*, pp. 295–314, 2008.

[4] S. Hou and P. D. Wentzell, "Re-centered kurtosis as a projection pursuit index for multivariate data analysis," *Journal of Chemometrics*, vol. 28, no. 5, pp. 370–384, 2014.

[5] M. E. Breaban, "Multiobjective projection pursuit for semisupervised feature extraction," in *Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16*, pp. 324–333, Springer, 2013.

[6] S. Espezua, E. Villanueva, and C. D. Maciel, "Towards an efficient genetic algorithm optimizer for sequential projection pursuit," *Neurocomputing*, vol. 123, pp. 40–48, 2014.

[7] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[8] B. Tran, B. Xue, and M. Zhang, "Variable-length particle swarm optimization for feature selection on high-dimensional classification," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 473–487, 2018.

[9] M. Labani, P. Moradi, and M. Jalili, "A multi-objective genetic algorithm for text feature selection using the relative discriminative criterion," *Expert Systems with Applications*, vol. 149, p. 113276, 2020.

[10] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings 6*, pp. 849–858, Springer, 2000.

[11] E. Zitzler, M. Laumanns, and L. Thiele, “Spea2: Improving the strength pareto evolutionary algorithm,” *TIK-report*, vol. 103, 2001.

[12] L. Abualigah and A. J. Dulaimi, “A novel feature selection method for data mining tasks using hybrid sine cosine algorithm and genetic algorithm,” *Cluster Computing*, vol. 24, pp. 2161–2176, 2021.

[13] R. Purushothaman, S. Rajagopalan, and G. Dhandapani, “Hybridizing gray wolf optimization (gwo) with grasshopper optimization algorithm (goa) for text feature selection and clustering,” *Applied Soft Computing*, vol. 96, p. 106651, 2020.

[14] L. M. Abualigah, A. T. Khader, and E. S. Hanandeh, “A new feature selection method to improve the document clustering using particle swarm optimization algorithm,” *Journal of Computational Science*, vol. 25, pp. 456–466, 2018.

[15] A. K. Abasi, A. T. Khader, M. A. Al-Betar, S. Naim, S. N. Makhadmeh, and Z. A. A. Alyasseri, “An improved text feature selection for clustering using binary grey wolf optimizer,” in *Proceedings of the 11th National Technical Seminar on Unmanned System Technology 2019: NUSYS’19*, pp. 503–516, Springer, 2021.

[16] A. K. Abasi, A. T. Khader, M. A. Al-Betar, S. Naim, S. N. Makhadmeh, and Z. A. A. Alyasseri, “A text feature selection technique based on binary multi-verse optimizer for text clustering,” in *2019 IEEE Jordan international joint conference on electrical engineering and information technology (JEEIT)*, pp. 1–6, IEEE, 2019.

[17] L. Abualigah, B. Alsalibi, M. Shehab, M. Alshinwan, A. M. Khasawneh, and H. Alabool, “A parallel hybrid krill herd algorithm for feature selection,” *International Journal of Machine Learning and Cybernetics*, vol. 12, pp. 783–806, 2021.

[18] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[19] B. Xue, M. Zhang, W. N. Browne, and X. Yao, “A survey on evolutionary computation approaches to feature selection,” *IEEE Transactions on evolutionary computation*, vol. 20, no. 4, pp. 606–626, 2015.

[20] S. Tabakhi, P. Moradi, and F. Akhlaghian, “An unsupervised feature selection algorithm based on ant colony optimization,” *Engineering Applications of Artificial Intelligence*, vol. 32, pp. 112–123, 2014.

[21] A. Gupta, I. S. Rajput, V. Jain, and S. Chaurasia, “Nsga-ii-xgb: Meta-heuristic feature selection with xgboost framework for diabetes prediction,” *Concurrency and Computation: Practice and Experience*, vol. 34, no. 21, p. e7123, 2022.

[22] H. B. Nguyen, B. Xue, P. Andreae, and M. Zhang, “Particle swarm optimisation with genetic operators for feature selection,” in *2017 IEEE Congress on Evolutionary*

Computation (CEC), pp. 286–293, IEEE, 2017.

[23] F. Tan, X. Fu, Y. Zhang, and A. G. Bourgeois, "A genetic algorithm-based method for feature subset selection," *Soft Computing*, vol. 12, pp. 111–120, 2008.

[24] A. K. Naik, V. Kuppili, and D. R. Edla, "Efficient feature selection using one-pass generalized classifier neural network and binary bat algorithm with a novel fitness function," *Soft Computing*, vol. 24, no. 6, pp. 4575–4587, 2020.

[25] P. Shamsinejadbabki and M. Saraee, "A new unsupervised feature selection method for text clustering based on genetic algorithms," *Journal of Intelligent Information Systems*, vol. 38, pp. 669–684, 2012.

[26] S.-S. Hong, W. Lee, and M.-M. Han, "The feature selection method based on genetic algorithm for efficient of text clustering and text classification," *International Journal of Advances in Soft Computing & Its Applications*, vol. 7, no. 1, 2015.

[27] N. Kushwaha and M. Pant, "Link based bpso for feature selection in big data text clustering," *Future generation computer systems*, vol. 82, pp. 190–199, 2018.

[28] B. Nakisa, M. Z. Ahmad Nazri, M. N. Rastgoo, and S. Abdullah, "A survey: Particle swarm optimization based algorithms to solve premature convergence problem," *Journal of Computer Science*, vol. 10, no. 9, pp. 1758–1765, 2014.

[29] Y. Lu, M. Liang, Z. Ye, and L. Cao, "Improved particle swarm optimization algorithm and its application in text feature selection," *Applied Soft Computing*, vol. 35, pp. 629–636, 2015.

[30] C. Paduraru, M.-C. Melemciuc, and A. Stefanescu, "A distributed implementation using apache spark of a genetic algorithm applied to test data generation," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1857–1863, 2017.

[31] Y. Vivek, V. Ravi, and P. R. Krishna, "Parallel bi-objective evolutionary algorithms for scalable feature subset selection via migration strategy under spark," *arXiv preprint arXiv:2205.09465*, 2022.

[32] F. Maqbool, S. Razzaq, J. Lehmann, and H. Jabeen, "Scalable distributed genetic algorithm using apache spark (s-ga)," in *Intelligent Computing Theories and Application: 15th International Conference, ICIC 2019, Nanchang, China, August 3–6, 2019, Proceedings, Part I 15*, pp. 424–435, Springer, 2019.

[33] H.-C. Lu, F. Hwang, and Y.-H. Huang, "Parallel and distributed architecture of genetic algorithm on apache hadoop and spark," *Applied Soft Computing*, vol. 95, p. 106497, 2020.

[34] C. Lam, *Hadoop in action*. Simon and Schuster, 2010.

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, *et al.*, "Spark:

Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[36] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba, "A study of master-slave approaches to parallelize nsga-ii," in *2008 IEEE international symposium on parallel and distributed processing*, pp. 1–8, IEEE, 2008.

[37] A. Verma, X. Llora, D. E. Goldberg, and R. H. Campbell, "Scaling genetic algorithms using mapreduce," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pp. 13–18, IEEE, 2009.

[38] C.-J. Ye and M.-X. Huang, "Multi-objective optimal power flow considering transient stability based on parallel nsga-ii," *IEEE Transactions on Power Systems*, vol. 30, no. 2, pp. 857–866, 2014.

[39] C. Hu, G. Ren, C. Liu, M. Li, and W. Jie, "A spark-based genetic algorithm for sensor placement in large scale drinking water distribution systems," *Cluster Computing*, vol. 20, pp. 1089–1099, 2017.

[40] L. Alterkawi and M. Migliavacca, "Parallelism and partitioning in large-scale gas using spark," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 736–744, 2019.

[41] V. S. Jonnalagadda, P. Srikanth, K. Thumati, S. H. Nallamala, and K. Dist, "A review study of apache spark in big data processing," *International Journal of Computer Science Trends and Technology (IJCST)*, vol. 4, no. 3, pp. 93–98, 2016.

[42] Z. Jiang, B. Gao, Y. He, Y. Han, P. Doyle, and Q. Zhu, "Text classification using novel term weighting scheme-based improved tf-idf for internet media reports," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–30, 2021.

[43] O. Roudenko and M. Schoenauer, "A steady performance stopping criterion for pareto-based evolutionary algorithms," in *6th International Multi-Objective Programming and Goal Programming Conference*, 2004.

[44] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, pp. 1188–1196, PMLR, 2014.

[45] L. Meng, R. Huang, and J. Gu, "A review of semantic similarity measures in wordnet," *International Journal of Hybrid Information Technology*, vol. 6, no. 1, pp. 1–12, 2013.

[46] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine learning*, vol. 42, pp. 143–175, 2001.

[47] C. Gatu, "Special Chapters on Artificial Intelligence - Lecture 2 (Probability and Statistics." `https://profs.info.uaic.ro/~cgatu/sc/`, 2023. [Online; accessed 31-March-2023].