2015 IEEE 18th International Conference on Computational Science and Engineering

A Parallel Computing Hybrid Approach for Feature Selection

Jorge Silva
Instituto de Telecomunicações & DCC,
Faculdade de Ciencias,
University of Porto, Portugal
Email: up201007483@alunos.dcc.fc.up.pt

Ana Aguiar
Instituto de Telecomunicações
Faculdade de Engenharia,
University of Porto, Portugal
Email: aaguiar@fe.up.pt

Fernando Silva
CRACS/INESCTEC,
Faculdade de Ciencias,
University of Porto, Portugal
Email: fds@dcc.fc.up.pt

***Abstract*—The ultimate goal of feature selection is to select the smallest subset of features that yields minimum generalization error from an original set of features. This effectively reduces the feature space, and thus the complexity of classifiers. Though several algorithms have been proposed, no single one outperforms all the other in all scenarios, and the problem is still an actively researched field. This paper proposes a new hybrid parallel approach to perform feature selection. The idea is to use a filter metric to reduce feature space, and then use an innovative wrapper method to search extensively for the best solution. The proposed strategy is implemented on a shared memory parallel environment to speedup the process. We evaluated its parallel performance using up to 32 cores and our results show 30 times gain in speed. To test the performance of feature selection we used five datasets from the well known NIPS challenge and were able to obtain an average score of 95.90% for all solutions.**

I. Introduction

In 2011, a report by McKinsey Global Institute asserted that machine learning is the key for innovation, competition, and productivity [1]. For several years machine learning has been widely studied, and new techniques and algorithms are constantly emerging. However, preparing a classifier for a classification task is not easy and researchers are commonly faced with difficulties such as: how much data is needed, what features should be added, and does the dataset has outliers and noisy data [2]. Usually, researchers gather as much information as possible about a problem and turn that information into a processed dataset for machine learning purposes. This methodology often leads to datasets with a large number of features, which in most cases means poor performance from the learning algorithm. The problem is commonly known as the curse of dimensionality [3]. Moreover, as more features are used the higher is the risk of overfitting, which means adapting a learning algorithm so much to the training data, that it starts *"memorizing"* examples instead of learning from them. Thus, drastically decreasing prediction accuracy for unseen data [3].

Feature selection is the process of selecting a subset of the original features so that the feature space is reduced according to a certain evaluation criteria [4]. The goal is to find the smallest subset possible that yields the minimum generalization error. There are several advantages of using feature selection: improving classification performance, reducing the time it takes to classify unseen data, and achieving a better understanding of the process that generates data [5]. Since feature selection is able to effectively reduce the dimension of the data, it is a commonly used technique to tackle the previous mentioned classification problems.

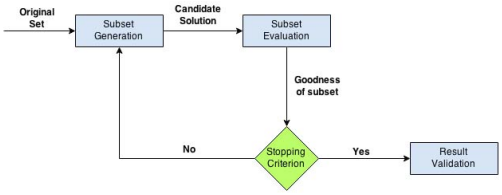

Fig. 1: The four key steps of feature selection.

Feature selection methods require a full search of the feature space, testing subsets of features, and evaluating them to find the final solution. The search space consists of the combination of all possible subsets, which for a dataset with n features produces a feature space of size 2^n. This makes an exhaustive search impracticable in most cases. For problems with a large number of features, finding an optimal subset of features is usually intractable and many problems of this kind are asserted as NP-hard [6]. Several algorithms exist in the literature that tackle this problem. Despite their differences they all follow the same general approach, which consists in four key steps: subset generation, evaluation of subset, stopping criterion, and result evaluation [4]. The first step defines how successors of a subset are generated and how the search is guided. The second step represents a function that is used to measure the quality of a subset. The conditions that make the search stop are defined in the stopping criterion. Finally, results validation is the process where the final solution from the feature selection algorithm is evaluated for its quality. Figure 1 illustrates an overview of the general process.

Depending on the size of the dataset and on the approach, feature selection algorithms can take significant time to reach the stopping conditions. Because of that, parallel computing emerges as an option to tackle this problem. Taking a closer look at the general procedure, the problem can be reformulated as a set of multiples tasks. On this scenario, a task could be defined as the process of generating a new subset, evaluating it, and checking if the stopping criterion is reached. Understandably, the processing of a task is completely independent of processing any other. This makes the feature selection problem an ideal candidate for parallelization.

978-1-4673-8297-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSE.2015.34

Commonly, feature selection algorithms need to compromise the goodness of their solutions in order to provide results in a practicable time. Moreover, wrapper strategies are known for producing the best results [6], however they are not usually used in high dimensional datasets because of their computational cost. This work proposes a new hybrid feature selection algorithm that uses a filter procedure to reduce the feature space and then uses a wrapper search implemented on a shared memory parallel environment to find the final solution. With this approach, we aim to achieve better solutions by using a more computationally expensive approach that explores more of the search space, combined with parallelism to speedup execution.

The remainder of the paper is structured as follows: next we present a brief review of the current state of art of feature selection algorithms. In section III we thoroughly explain each component of our approach and how they act together. Section IV details the proposed novel heuristics applied in our wrapper search component. Section V details the implementation of our strategy on a shared memory parallel architecture. Section VI assesses the parallel performance of the implemented algorithm. Section VII empirically evaluates and discusses the results attained with our strategy on several public datasets. Finally, the last section discusses future work and present conclusions on our work.

II. State of Art

Feature selection has been widely studied and as result a large number of algorithms have been proposed. These algorithms can be categorized into three groups: filter, wrapper, and embedded [4]. Filter algorithms use a classifier independent metric to evaluate either individual features or subsets. The idea is to identify which features are more relevant to the learning task. These methods assume complete independence between data and the learning algorithm. As a result, the final solution could be applied to several learning algorithms without the need to run the filter algorithm more than once. Usually the metric is fast to compute, therefore filter methods have low-computational cost. However, in most cases they fail to produce the optimal subset of features and usually perform worst than other types of feature selection algorithms. Examples of filter algorithms in literature are found in [7], [2], [6], [5].

Wrapper algorithms find the final solution using a learning algorithm as part of the evaluation criteria. The main idea of these methods is to use the learning algorithm as a *"black-box"* to guide the search for the optimal solution. The learning is applied to every candidate solution and the goodness of the subset is given according to the performance of the learning algorithm. Due to the learning algorithm being directly used on the process of selecting features, these methods tend to find better solutions. Nonetheless, the final solution only applies for the selected learning algorithm, since using a different one will most likely result on a different final solution. These methods have higher computational cost as they require training and classifying data for each candidate solution. Moreover, cross-validation techniques are commonly used, which further increases the computational cost of the algorithm [8]. Combining different search strategies with different classification algorithms results in a new wrapper method, and several examples can be found in the literature [9], [7], [10].

Embedded methods are inspired by wrapper and filter algorithms and try to use the best qualities from both types. These algorithms encapsulate feature selection with classifier construction. By doing that, the feature selection part interacts with the learning algorithm. However, it does not require training the classifier and thus they are usually faster than wrapper methods. Since these methods do not separate feature selection from learning, they are very specific to a learning algorithm. Meaning that an embedded method can only be applied to a specific learning algorithm. Tang et. al. [11] categorizes embedded methods into three groups and provides examples of algorithms for each type.

A. Hybrid Methods

Approaches that combine two categories of feature selection algorithms are gaining importance in the community. They are called hybrid methods and combine filter and wrapper methods in order to further improve the feature selection process. The idea behind these methods is to use a filter method to cut the search space into a smaller space, and then use a wrapper method to select the final solution. As examples of hybrid algorithms, we have the IFSFF algorithm [12] which uses a filter method to rank features in order to guide more efficiently the wrapper search. Another example is the Quick Branch and Bound algorithm which uses a filter approach to define subsets as starting points for a wrapper algorithm [7]. More hybrid algorithms are described in [6].

B. Parallel Feature Selection

Selecting the ideal set of features is far from an easy task. It usually requires many attempts until the desired result is attained. A conventional methodology is to change parameters on the algorithms or test different algorithms to compare results. Moreover, depending on the size of the dataset and on the algorithm chosen, a feature selection process can take a large amount of time. This triggered researchers to exploit parallelism within feature selection algorithms in order to improve their executions times. For example, Azmandian et. al. [13] used GPUs to accelerate their feature selection algorithm. Li et. al. [14] also resorted to parallelism to speed up a genetic search in the context of feature selection.

III. Overview of the Proposed Algorithm

In this section, we introduce our proposed hybrid method. It starts with a filter approach that ranks features individually. Based on a threshold and on the calculated rank, features are selected to the next phase. The goal of the filter is to use a less costly computational method to reduce the search space. Therefore, removed features are considered irrelevant and are not used any further in the next stages of the algorithm. We use Mutual Information (MI) [5] as the metric to individually rank features. The wrapper phase searches the feature space by using a novel meta-heuristic in order to find the final subset of features. Wrapper methods use a learning algorithm to evaluate the goodness of a subset. In our approach we use Support Vector Machines (SVM) [15] as our learning algorithm.

The filter and wrapper components represent the main functions of the algorithm and are responsible for selection

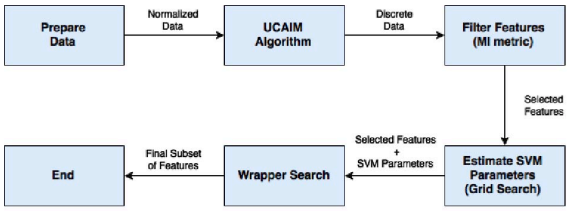

Fig. 2: Workflow of the proposed method.

features. Because, the wrapper search is the most significant contribution of this paper, we dedicate a full section to it (Section IV). Both components require some pre-processing steps that are executed by two additional algorithms: Uncertain Class Attribute Interdependency Maximization (UCAIM) [16] and Grid Search [17]. The first one is used to discretize data, which is a recommended step when using SVMs, and it is a mandatory procedure to calculate MI in cases where variables have continuous values. The Grid Search is a very popular procedure used to estimate the parameters of learning algorithms.

The workflow of our proposed method is illustrated in figure 2. We start by preparing data, then discretize it with UCAIM algorithm. Then, the feature space is reduced by eliminating features that are not able to pass the MI filter. The next step is to estimate the SVM parameters using the grid search. Finally, the algorithm runs the wrapper search which is responsible to find the subset of features that is presented as final solution. All algorithm steps are explained in more detail in the following sections.

A. Prepare Data

This is the stage where data is read from files and pre-processed. In most cases, pre-processing data includes techniques to find outliers that may jeopardize the performance of the learning algorithm. Although there are several techniques to detect and remove outliers, this process usually requires some knowledge about the dataset. This procedure is rather specific to the dataset and thus we do not include it as part of our method. Instead, we assume that the dataset is already clean and ready for the algorithm. In any case, this stage implements normalization of the feature values to a scale from 0 to 1. This is a recommended procedure in order to improve the performance of learning algorithms [2].

B. UCAIM Algorithm

In order to discretize data, we selected the UCAIM algorithm, which is an evolution of the original CAIM algorithm. Both methods have the goal to delineate intervals on data in such a way that the interdependence between features values and class labels is maximum. Despite the fact that both algorithms perform well, the evolutionary approach adds the offset component, which takes into account cases where data is unbalanced. The UCAIM algorithm has been shown to outperform the original one [16].

The UCAIM algorithm starts by setting the initial discretization scheme, D, as a set of two elements: the maximum and minimum values. Then, it proceeds to define a set of possible points. These are all the midpoints between each adjacent pair in the sorted and non-duplicate set of values. After that, UCAIM iteratively tries to add possible points to D. At each round, all possible points are added, one at the time, to D. Then, formula 1, which tries to maximize the interdependence between classes, is used to evaluate the quality of D with the recently added point. At the end of the round, the point with the best score is definitely appended to D. The process stops, when no point could improve the score that D has at the start of the round. By the end of the UCAIM algorithm, we get a discretization scheme D. Later, for each feature value, we discover the interval on D where it belongs, and convert the value to the midpoint of that interval. Thus, achieving the desired discrete data.

Algorithm 1 illustrates the steps needed to find D for a given feature F_i and its possible values V_i on a classification problem with S label classes.

$$UCAIM(F_i, D) = \frac{\sum_{r=1}^{n} \frac{max_r^2 \times Offset_r}{M_{+r}}}{n} \quad (1)$$

Where n is the number of intervals, r iterates through all intervals, max_r is the maximum value inside an interval, M_{+r} is the total number of values on the interval and offset:

$$Offset_r = \frac{\sum_{i=1}^{S} (max_r - q_{ir})}{S - 1} \quad (2)$$

where S represents the classes labels, q_{ir} are the number of values in interval r that belong to class i, and max_r is the maximum number of values in interval r across all classes. Basically, $Offset_r$ is the average difference of the number of points in all classes to the number of points in the class that has the most points in that interval.

Algorithm 1 UCAIM Algorithm

```
procedure UCAIM(V_i, S)
    values ← REMOVEDUPLICATES(V_i)
    min, max ← FINDLIMITS(values)
    B ← GENERATEPOSSIBLEPOINTS(values)
    K ← 1, D ← {min, max},
    BestS ← 0, BestP ← {}
    while K ≤ S or GlobalUCAIM < BestS do
        GlobalUCAIM ← BestS
        D ← D ∪ BestP
        for P ∈ B do
            auxD ← D ∪ P
            auxS ← GETUCAIMSCORE(auxD)
            BestS, BestP ← UPDATEBEST(P, auxS)
        K ← K + 1
```

C. Filter Metric

In contrast to some feature selection algorithms, we do not intend to use a filter approach to find a final subset of features. Instead, our method uses it as a pre-processing step to eliminate features and make it practicable for a more intensive

search on the wrapper part. Therefore, our filter should have the following characteristics:

1) **Evaluate single features.** Several filter approaches evaluate subsets of features. However, to keep a low computational cost, we avoid searching for feature subsets and evaluate features only individually.
2) **Not very restrictive.** The percentage of removed features should not be very large. Although as less features pass the filter the faster the wrapper ends, it is difficult to accurately assess the quality of a feature just by using a filter metric. It has been shown that features considered irrelevant when individually evaluated, are in fact important when inserted into a specific set of features [2]. Hence, to avoid compromising the performance of the final solution, it is important to avoid removing a large number of features at this stage.

There are several algorithms in the literature that fulfil the first requirement of our list, these methods are called univariate [12]. Two of the most commonly used metrics of this type are Mutual Information (MI) and Pearson Correlation Coefficients (PCC) [2]. Both metrics measure the dependence between two variables. Nonetheless, there is a key difference between them. MI measures the general dependence between the variables while PCC measures linear dependence. Li et al. [18] tested this property and concluded that this makes MI a better metric. Based on that work and on the amount of other works that use MI [5], we decided to use it as the selected metric to our filter approach.

Calculating the MI score for every feature does not remove features by itself, so in the next step we define a strategy that filters features taking into account the required second characteristic. The idea is to define a threshold and to remove features for which the MI score is bellow that threshold. Since MI scores diverge a lot when changing datasets, it is not possible to use a fixed threshold. Instead, we define the threshold as a percentage of the maximum MI score, and leave out features with MI below the threshold.

D. Grid Search

As previously mentioned, we use the SVM as our learning algorithm. As we will show in the next sections, SVMs have some parameters that must be tuned in order to provide better results. However, it is not uncommon for researchers to not knowing which parameters to use. On our method, we let users define the parameters; yet, if they do not specify them, the algorithm estimates the best parameter set to use. We test several parameters and select the ones that provide the best results using a grid search. This method tests parameters in two ranges. First, a large scale range and then, after choosing one value for the larger range, a smaller scale range is used. For example, suppose that for a parameter i the first possible values are $L_i = \{..., 2^7, 2^9, 2^{11}, 2^{13}, ...\}$. Now imagine that the selected value from the L_i is 2^9, hence the algorithm proceeds to search the final parameter in the following range $S_i = \{..., 2^{8.5}, 2^{8.75}, 2^9, 2^{9.25}, 2^{9.5}, ...\}$.

Typically grid search is applied to tune the classifier using the set of features. However, because we use the classifier to select a subset of features, we need to choose the parameters before knowing the feature set to be used. In addition, it is impracticable to perform a grid search for every subset being evaluated during the wrapper search. Thus, we perform a grid search on n randomly generated feature subsets after the filter and before starting the wrapper search. The best set of parameters for each feature subset counts as a vote, and the parameter set with most votes is selected. In cases where the highest number of votes is the same for more than one set of parameters, we generate n new random subsets and the test is repeated for the tied set of parameters. This process is repeated until there are no more ties.

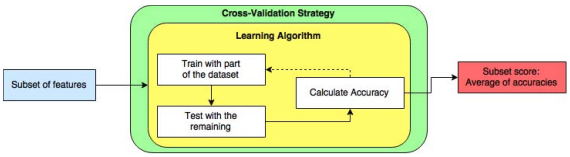

Fig. 3: Subset Evaluation

IV. Wrapper Search

Our proposed feature selection algorithm was designed to make use of existing algorithms for most of the tasks that must be performed. However, the wrapper search is a new meta-heuristic which, together with the strategy for its parallel execution, makes it a main contribution of this work. This is the most complex part of our algorithm and the functions used at this stage define its computational cost and its ability to find good solutions. For the sake of understanding, we further divided its explanation into three sections: learning algorithm, search strategy, successor generation. The following section complements the description with the parallel strategy.

A. Learning Algorithm

We use SVM as the learning algorithm to our method based on the work [15] in which the authors compare several learning algorithms in the context of classification problems. They concluded that SVM in general outperforms other classifiers. SVMs can have different kernels, and each one defines a distinct way to map data into higher dimensions. In order to select an adequate kernel, size and type of data should be taken into account [17]. The number of parameters to be estimated also depends on the kernel selection.

The function of the SVM in our approach is to evaluate the goodness of a subset of features. For each tested subset, we train the classifier with a part of the data and test it with the remaining. This process is commonly known as cross-validation [17] and the number of times it is performed for a subset depends on the defined number of folds. In the end of the whole process the algorithm gets a score for the subset. This score is the average of the accuracy obtained for each fold. This process is illustrated by figure 3.

B. Search strategy

Although several search strategies exist and have been used on wrapper approaches, we decided to implement a new meta-heuristic that explores the search space more thoroughly. The idea was to create a different strategy to search for solutions,

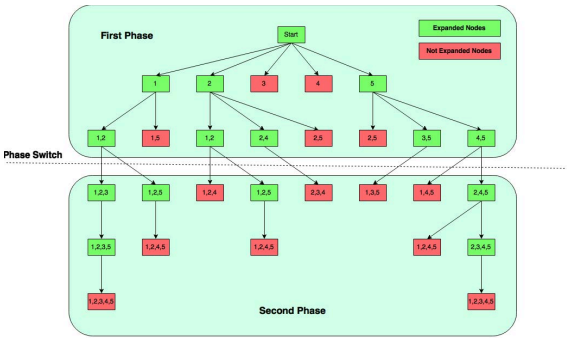

Fig. 4: Example of the implemented wrapper search

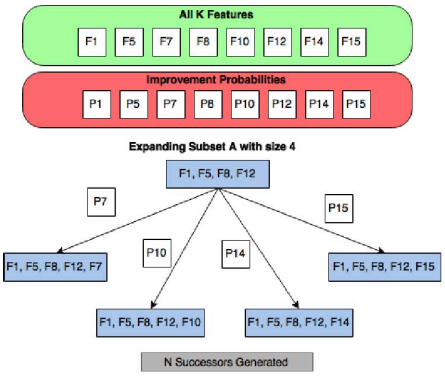

Fig. 5: Successor Generation using improvement probabilities

while maintaining a structure that could be easily run with multiple processors. The proposed search organizes subsets as nodes on a tree whose first level is composed by n starting subsets, each with a single feature not removed by the filter. From now on, we use subsets and nodes interchangeably.

The innovative idea of the proposed search is to explore broadly different regions of the search space, looking for the areas of higher classification accuracy, and then focus on searching the local maxima in each region. Thus, the search strategy doesn't have an uniform behaviour, but is divided into two stages. First, we gather as many good solutions as possible. Then, we improve them up to the best score they can reach on the second stage. The transition between stages takes place when subsets reach a certain number of features. Figure 4 illustrates an example of the implemented wrapper search.

In the first stage, the decision to expand a node or not is based on the distance from the obtained score to the global best. In this step, a threshold is defined and nodes whose difference of score to the global best is lower than the threshold are expanded further. During the second stage, nodes are searched using a depth first strategy and they are expanded while they still improve the score of their parent. The search stops when there are no more nodes left to explore. In this stage, a mechanism cuts subsets with a certain probability to avoid excessive work. The probability of each subset being cut is defined based on how distant its score is from the global best, according to the following table:

% Distance to global	Cut probability
$d < 0.5$	0%
$0.5 \leq d < 1.0$	25%
$1.0 \leq d < 1.5$	50%
$d \geq 1.5$	75%

The mechanism executes every t seconds, where t is a value which can be user defined.

The defined threshold in the first stage and the size at which stages switch have a great impact on the amount of nodes explored in the search. Thus, it is possible to define how restrictive the search is by manipulating these parameters.

C. Successor Generator

Finally, we explain how to generate successors of a subset. The idea is to add to a subset S_j several features that are not yet part of it. Each feature can be added with a specific probability, according to a likelihood of improving the evaluation score, estimated in a pre-processing step described below. If k features are selected to be added, then k new successors of S_j are generated. Each one represents S_j combined with a new feature. By doing so, our method increases the likelihood of features with high improvement score being added to new subsets. Figure 5 illustrates this process.

The likelihood of a certain feature contributing to an improvement in the evaluation criteria is estimated in a pre-search phase. The procedure starts by generating n random subsets, and evaluates each one of them using the subset evaluation procedure. Then, for every F_i we test how the score of a subset improves when F_i is either added or removed. In the end we count how many times F_i improved subsets to calculate the likelihood of improvement. These values are then used when the wrapper search decides to expand a node.

It is clear that our successor generator function may generate repeated subsets. Since the function that evaluates subsets is deterministic, testing a subset more than once is a waste of computations. But the search strategy does not handle the problem. Thus, we added a mechanism to avoid work repetition to the process of generating successors. The hash value of every tested subset is added to an hash table. Then, every time a new subset is generated, a look up in an hash table checks whether it has been tested before, if the answer is positive, then the successor is discarded.

D. Overview of the wrapper search

In the previous sections, we discussed the several components of the proposed wrapper search. It is also important to understand how they act together. Algorithm 2 details the overall wrapper search strategy.

V. Parallelized Method

The UCAIM, Filter and Grid Search parts of our proposed approach are not computationally costly, however, the wrapper part is. Our search strategy is very intensive in the number of

Algorithm 2 Search Strategy

```
1:  procedure SEARCH(size, W, data, hashTable, probs )
2:      lastStage ← False
3:      while W ≠ empty do
4:          s ← REMOVELAST(W)
5:          if SIZE(s) ≥ size then
6:              lastStage ← True
7:          score ← SVMCLASSIFICATION(s, data)
8:          if WORTHEXPAND(score, s, lastStage) then
9:              newN ← EXPAND(s, hashTable, probs)
10:             UPDATEGLOBAL(score)
11:             if lastStage then
12:                 W ← W ∪ newN
13:             else
14:                 W ← newN ∪ W
```

explored nodes. Thus, finding a way to realise the search in parallel will obviously help in reducing the computation time. Although the first three parts are not very expensive, we have also parallelized them. On UCAIM and Filter parts, our method achieves parallelism by dividing features by processors. By contrast, on the Grid Search the random generated subsets are divided by the processors. Since in all parts, problems were divided into smaller tasks and each one of them is independent, this presented no major difficulty.

On the other hand, the wrapper presents a challenge that can compromise the performance of the algorithm. The wrapper requires some information such as the hash table and best score to be global accessible and constantly updated. Keeping such structures always updated when executing with multiple processes may become computationally costly in terms of performance.

A. Parallel Scheme

We can visualise the problem of our wrapper search as a set of tasks. Each task is the process of getting a subset from a work list, evaluate it, decide to either expand it or not, and in the case of expanding, generate new subsets, remove those that have been already been tested and in the end, add all the remaining ones to the work list.

The idea of our parallel wrapper is to define local work lists on every process and use them to store subsets that still have to be processed. Then, iteratively, having each process computing a task for a subset. The processes would get the subset from their local work list and add new work there as well. Additionally, in order to remove new generated subsets already tested, a global hash table that is accessible by all process is used. There, the hash value of every tested subset is stored.

At the beginning of the search, the individual subsets of features are equally distributed among all processes and each one starts the computations as previous described. During the search, some processes will run out of work while there are others with a lot of work left. In order to provide a good work balance, a process P_i that hasn't any work, is able to request it from other process P_j. When P_j sees the request, it will send part of its work back to P_i.

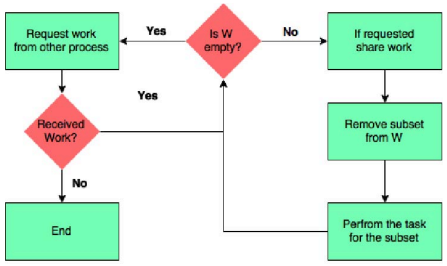

Fig. 6: Workflow of a single process

The overall workflow for a single process is illustrated by figure 6, where W represents the local work list.

B. Global information

To achieve the desired parallelism, we decided to use a shared memory environment where processes have access to the same memory addresses simultaneously. This paradigm provides a cheap way of communication between processes and avoids redundant copies of informations across multiple processes.

Global information is required to keep an hash table updated as well as to store the best score found during the search. To access and update this information we had to use different strategies. The best score is a single variable defined in shared memory. The access to it is controlled by a mutual exclusion mechanism to avoid race conditions.

The hash table required some additional work, using a unique hash table and have every process constantly updating it and searching on it for repetitions is not an option. Mainly because processes would start writing in the same memory spaces. Thus, the hash table would become incoherent, probably leading to the loss of entries which would result in lots of repeated work. On the other hand, using a mutual exclusion mechanism is not viable because waiting for the access to the memory would drastically decrease the parallel performance.

In order to make this work we divide the hash table into p partitions, each one is assigned to a process P_i. Only process P_i is allowed to write in the partition assigned to it. However, any process is free to read from any partition at any given time. By doing that, each process stores the hash of its tested subsets in its own partition. Then, when it has to check if a new generated subset is new, it can read from every partition without having to wait for permission. Mutual exclusion was avoided and memory coherence is guaranteed because for every single memory address, only one process is allowed to write on it.

VI. Performance of the Parallelization

In this section we assess the performance of the implemented parallel search with the increase in the number processes used in the computation. It is common to run the same program several times using a different number of processing units and then get the execution times of each one of them. However, in our case, changing the number of processes also changes the amount of visited nodes. For this reason and to

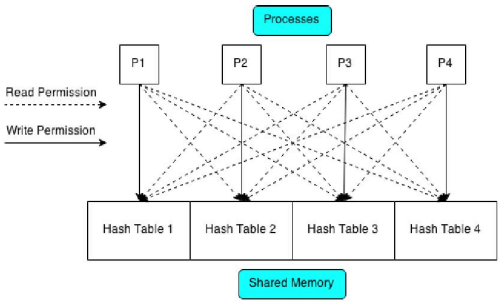

Fig. 7: Hash scheme in shared memory.

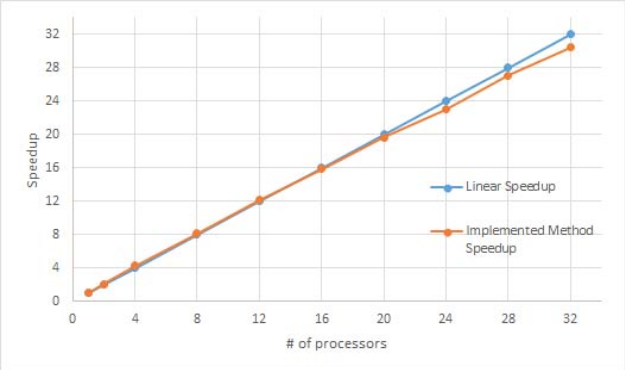

Fig. 8: Comparing Speedup against linear

perform all tests with the same conditions, we decided to test the worst case scenario when 20 features reach the wrapper search. The worst case scenario for our search is defined by the following properties:

1) Every subset is expanded
2) When expanded, each subset is combined with all possible features
3) The cutting probability for all subsets is 0%.

More accurately, this results in an exhaustive search with a very large number of attempts resulting from repeated work. Also, this means that regardless of the number of processors, all tests would search the same feature space which is a total of $2^{20} - 1 = 1048575$ subsets. Table I illustrates the results of our test and figure 8 compares the obtained speedup against the linear.

TABLE I: 20 Feature Exhaustive Search Data

# Processors	Execution Time (sec)	Speedup	Number of tests
1	133984	1.00	1048575
2	64148	2.09	1048575
4	31427	4.26	1048575
8	16406	8.17	1048575
12	11055	12.12	1048576
16	8457	15.84	1048575
20	6812	19.67	1048575
24	5825	23.00	1048578
28	4948	27.08	1048581
32	4409	30.39	1048579

The results show an almost linear speedup in performance of the parallel search when 32 cores are used. In some cases, the implemented strategy was able to achieve speedups greater than the ideal values. Moreover, it was able to efficiently avoid repeated work by having every process keeping track of their tested subsets in its partitioned hash table and checking all the partitioned hash to avoid repeating them. In the worst case, only 6 subsets out of the 1048575 total subsets were tested more than once.

VII. Feature Selection Results

In 2003, the NIPS[19] feature selection challenge was created and its aim was to find which algorithm would preform better in terms of classification. The contest consists of five public datasets, each divided into three sets: train, validation, and test. For the first two, it is possible to access both data and labels, while on the last one only data is available. The idea of the challenge is to use feature selection and machine learning techniques on the train and validation sets in order to construct a classifier that is able to accurately predict the labels of the test set. At the end of the process, one can submit the generated predictions on the website and it provides feedback about the results. Nowadays the challenge is still open and it allows researchers to benchmark their systems. Thus, we decided to use it in order to test the performance of our approach. The following table presents the characteristics of the five datasets[19]:

Dataset	Type	Features	Train Examples	Validation Examples	Test Examples
Arcene	Dense	10000	100	100	700
Dexter	Dense	5000	6000	1000	6500
Gisette	Sparse integer	20000	300	300	2000
Dorothea	Sparse binary	100000	800	350	800
Madelon	Dense	500	2000	600	1800

Regarding user defined parameters, we used 0.5 for the search threshold, 8 for the size at which the search changes, and 900 seconds for the cutting mechanism. In addition to that, we executed each test using 62 cores and RBF as the SVM's kernel. These were the fixed values for all tests, however some parameters such as cross-validation technique and percentage of the filter had to be adapted to each dataset. The following table presents the used parameters and the results obtained:

Dataset	Filter Threshold	Features Post-Filter	Cross-Validation	Size Final Subset	Final Score	Time (sec)
Arcene	0.50	232	Leave-one-out	14	99.00	2156
Dexter	0.94	364	Leave-one-out	72	98.50	32539
Gisette	0.97	121	5 folds	85	97.32	53560
Dorothea	0.90	450	5 folds	30	97.63	32962
Madelon	0.97	255	10 folds	14	87.10	33117

After obtaining a final subset for each dataset, we used it to train a classifier only using data from the train set. Then, we predicted the labels for every one of the three sets: train, validation, and test. Later, results were submitted to the NIPS website and the accuracy of our predictions as well as the rank among all the submissions are illustrated on the following table:

The results we obtained ranked among the top 60% for each dataset. This outcome came at no surprise considering

Dataset	Accuracy Train	Accuracy Validation	Accuracy Test	Rank
Arcene	99.00	82.00	74.56	892/1503
Dexter	98.33	83.67	81.65	819/1007
Gisette	98.97	96.80	96.67	465/932
Dorothea	95.63	94.29	77.18	475/812
Madelon	93.35	87.50	88.67	344/1059

that the goal of the challenge set used is directed to evaluate the overall performance of the classification system. Despite being part of the machine learning workflow, feature selection is not the whole process and usually several more techniques such as outlier detection, noisy examples removal and generation of synthetic data are required [20]. Moreover, knowledge about the specific problem at hand can improve the generalisation result by targeting feature choice, or through the use of another metric for calculating the feature subset score in the search. In our experiments, we focused on testing the ability of our search algorithm to find what was defined as a good subset according to the score, which was the accuracy of the learning algorithm on the training sets. Although cross-validation strategies were used to improve generalization, they were not enough, and in general the classifier was not able to predict well on unseen data. Nevertheless, we are quite happy with the results on the NIPS challenge, because we could confirm that our hybrid approach, without any further analysis of the dataset nor additional techniques, was able by itself to produce quite acceptable results.

In terms of feature selection, our approach, in most cases, was able to produce a final subset of features that was much smaller than the original set of features and that had very high accuracy score. Although we cannot guarantee that the optimal subset was found, it would require to test the whole search space, our algorithm was able to achieve results near the perfect score in 4 out of the 5 tests. Moreover, the results were obtained in a practicable amount of time, considering the size of the datasets and the cross-validations methods used.

VIII. Conclusion

In this paper, we propose a new hybrid feature selection approach that resorts to a novel parallel search strategy to speedup execution. It starts by reducing the feature space using a filter and then uses a wrapper method to find the final solution. Because of their high-computation cost, wrapper methods are not commonly used on high dimensional datasets. In our experimental evaluation, we tested high dimensional datasets, thus showing how it is possible to take advantage of parallelism to thoroughly search larger spaces in a practicable amount of time. Our initial results show an almost linear speedup up to 32 cores while being able to find solutions with near perfect score.

We are aware that the accuracy of the classifier can be improved by employing better generalization methods, but that was not the goal of the task at hand. We intend to experiment on more problems and with other filter methods, namely PCC [2] and Relief [4], as well as with other classifiers Decision Trees and kNN[20], methods that fit well with the approach we proposed.

Acknowledgment

We would like to thank Isabelle Guyon and Lukasz Romaszko who kindly reopening the NIPS 2003 challenge.

This work was partially supported by national funds through project VOCE (PTDC/EEA-ELC/121018/2010), and in the scope of R&D Unit UID/EEA/50008/2013, funded by FCT/MEC through national funds and when applicable co-funded by FEDER/PT2020 partnership agreement.

References

[1] J. Manyika and et. al., "Big data: The next frontier for innovation, competition, and productivity," *McKinsey Global Institute*, no. May, p. 156, 2011.

[2] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *J. Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[3] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[4] V. Kumar and S. Minz, "Feature Selection: A literature Review," *Smart CR*, vol. 4, no. 3, pp. 211–229, 2014.

[5] J. R. Vergara and P. A. Estévez, "A review of feature selection methods based on mutual information," *Neural Computing and Applications*, vol. 24, no. 1, pp. 175–186, 2013.

[6] H. Liu and L. Yu, "Toward Integrating Feature Selection Algorithms for Classification and Clustering," *IEEE Trans. Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[7] L. Molina, L. Belanche, and A. Nebot, "Feature selection algorithms: A survey and experimental evaluation," in *ICDM'2002*, 2002, pp. 306–313.

[8] K. Dunne, P. Cunningham, and F. Azuaje, "Solutions to Instability Problems with Sequential Wrapper-based Approaches to Feature Selection," *J. Machine Learning Research*, pp. 1–22, 2002.

[9] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[10] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[11] J. Tang, S. Alelyani, and H. Liu, "Feature Selection for Classification: A Review," in *Data Classification: Algorithms and Applications*, C. Aggarwal, Ed. CRC Press, 2014.

[12] J. Xie and W. Xie, "A Novel Hybrid Feature Selection Method Based on IFSFFS and SVM for the Diagnosis of Erythemato-Squamous Diseases," *JMLR Workshop on Applications of Pattern Analysis*, vol. 11, pp. 142–151, 2010.

[13] F. Azmandian, A. Yilmazer, J. G. Dy, J. A. Aslam, and D. R. Kaeli, "GPU-Accelerated Feature Selection for Outlier Detection Using the Local Kernel Density Ratio," in *ICDM'2012*, 2012, pp. 51–60.

[14] R. Li, J. Lu, Y. Zhang, and T. Zhao, "Dynamic Adaboost learning with feature selection based on parallel genetic algorithm for image annotation," *Knowl.-Based Syst.*, vol. 23, no. 3, pp. 195–201, 2010.

[15] V. G. C., "Performance Evaluation of Machine Learning Classifiers in Sentiment Mining," *Int. Journal of Computer Trends and Technology (IJCTT)*, vol. 4, no. 6, pp. 1783–1786, 2013.

[16] J. Ge, Y. Xia, and Y. Tu, "A Discretization Algorithm for Uncertain Data," in *DEXA'2010, LNCS 6262*, 2010, pp. 485–499.

[17] C. Hsu, C. Chang, and C. Lin, "A Practical Guide to Support Vector Classification," *BJU international*, vol. 101, no. 1, pp. 1396–400, 2008.

[18] W. Li, "Mutual information functions versus correlation functions," *Journal of Statistical Physics*, vol. 60, no. 5-6, pp. 823–837, 1990.

[19] "NIPS 2003 feature selection challenge," http://www.nipsfsc.ecs.soton.ac.uk/results/?ds=overall, accessed: 2015-04-23.

[20] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica (Slovenia)*, vol. 31, no. 3, pp. 249–268, 2007.