A Single Program Multiple Data Algorithm for Feature Selection

Bhabesh Chanduka, Tushaar Gangavarapu, Jaidhar C. D.

Department of Information Technology
National Institute of Technology Karnataka, Surathkal, India, 575 025
{bhabesh.chanduka, tushaargvsg45}@gmail.com
jaidharcd@nitk.edu.in

Abstract. Feature selection is a critical component in data science and has been the topic of research for many years. Advances in hardware and the availability of better multiprocessing platforms have enabled parallel computing to reach very high levels of performance. Minimum Redundancy Maximum Relevance (mRMR) is a powerful feature selection technique used in many applications. In this paper, we present a novel optimized Single Program Multiple Data (SPMD) approach to implement the mRMR algorithm with synchronous computation, optimum load balancing and greater speedup than task-parallel approaches. The experimental results presented using multiple synthesized datasets prove the efficiency and scalability of the proposed technique over original mRMR.

Keywords: Feature Selection, Parallel, SPMD, mRMR

1 Introduction

Feature selection is one of the key techniques to filter information for data analysis [1–6]. It is the process of selecting a subset of crucial features from a given set of features, for efficient model construction. It is needed for high-dimensional data for mainly two reasons. Firstly, it reduces the training time which could have exponential complexity in the number of features. Secondly, it reduces the risk of over-fitting.

Feature selection techniques can be classified into three types: filter, wrapper [7] and embedded methods. Lot of research has gone into developing efficient feature selection techniques for optimizing space and time complexity. The earliest techniques used were the exhaustive and branch and bound techniques [8, 9] which had exponential complexity in the number of features. Improvements included genetic algorithms [10] and particle swarm optimization [11], but even though they reduced the execution time complexity, they provided only local optimums. To overcome this, sequential and incremental feature selection techniques were adopted [1, 12]. These techniques drastically reduced the complexity to linear from quadratic. To further improve these, parallel and distributed computing techniques [13, 14] were used.

This paper aims at improving the minimum Redundancy Maximum Relevance (mRMR) [1, 15] feature selection technique by implementing the algorithm in parallel to further boost its efficacy. This allows it to scale well with high-dimensional datasets having large number of samples and is particularly useful for areas like genome processing, financial data and computer vision where datasets have both large number of samples and features. The rest of the paper is organized as follows. In Section 2 we discuss approaches that have been explored for improving the mRMR feature selection algorithm. Section 3 describes in detail the approach for implementing a data-parallel version of the mRMR methodology. In Section 4 we discuss the results of our research followed by conclusions and future work in Section 5.

2 Related Work

Multiple feature selection techniques have been developed, explored, optimized and improvised using task parallel approaches to improve the performance of machine learning algorithms. Peng et al. developed mRMR, an embedded technique to capture the relevance and redundancy [1, 15]. Since its development, mRMR has been used as a potential feature selection technique, thus increasing the need of its optimization and improvisation. Alshamlan, H et al. [22] used artificial bee-colony (ABC) swarm intelligence approach to analyze the microarray gene expression. They evaluated mRMR-ABC optimization against mRMR with genetic algorithms and with particle swarm optimization to establish its potential. Alomari et al. [25], used mRMR with Bat algorithm to identify most important and informative gene. Kaya, H et al. [24] used random forests in laughter detection and established that mRMR significantly reduces dimensions without affecting the performance. Enireddy, V et al. [23] extracted best texture and shape features using mRMR and fisher technique. The above presented works establish mRMR as a potential feature selection technique and the following works show the effectiveness of parallelization in feature selection.

A parallel course-grained version of the FortalFS feature selection technique was presented by de Souza J. T. et al. [14]. They also showed how master-slave parallel model can help achieve greater performance. Zhao Li et al. [26] presented MapReduce based parallelization for efficiency and scalability of maximum mutual information based feature selection. General theory that provides sufficient conditions that help correctly identify true features was developed by Zhou Y et al. [27]. The following works present the approaches taken to optimize and parallelize the mRMR feature selection algorithm.

A task-parallel approach for a slightly modified version of the mRMR algorithm was devised by LeKhac N. et al. [13] with emphasis on parallelizing the cross-validation step to select the optimal number of features that best represent the entire feature set. An Information Theory based approach was investigated by Ramrez Gallego et al. [16] to explore an efficient implementation of the mRMR feature selection technique over distributed systems and the method scaled well with high-dimensional datasets with a large number of samples. In [17], Ramrez

Gallego et al. proposed an improved version of the mRMR algorithm by optimizing redundancy calculations and presented a sequential implementation of the same in C++, a parallel version in CUDA for GPUs and a distributed version in Scala for Apache Spark framework. Their proposed method successfully optimized the data-access pattern held by the original algorithm and transformed the original quadratic complexity of the mRMR algorithm into an efficient greedy process. In [18], Claudio Reggiani et al. studied the design and scalability of the mRMR algorithm in MapReduce and also investigated its performance over dense as well as sparse data. It provided an improvement over the approach by Ramrez Gallego et al. [16] where the distributed method provided favorable methods for both traditional and alternative encoding, with the possibility of customizing the feature score function. In this paper, we propose a data parallel approach to improve the efficiency of the mRMR feature selection algorithm and also experimentally establish the scalability of the proposed approach.

3 Methodology

This section first describes the sequential mRMR approach [1] along with the discussion of some relevant parallel computing considerations. This is followed by the detailed algorithm to implement the mRMR in parallel using the SPMD approach.

3.1 The mRMR Approach

In terms of mutual information (I), feature selection serves to find a feature subset (S) with (m) features $\{x_i\}$ from M features, which collectively have largest dependency on the target class (C). This is called Max-Dependency, and takes the following representation:

$$max\, D(S, C),\, D = I(\{x_i, i = 1, 2, 3, ..., m\}, C) \tag{1}$$

The above condition 1 is hard to implement. For this reason, the original mRMR resorted to using Max-Relevance instead to select the best subset of representative features. This aims to search for features satisfying 1, to approximate $D(S, C)$ in 2 with the average mutual information values of the individual feature x_i and class (C):

$$max\, D(S, C),\, D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i, C) \tag{2}$$

Although this condition chooses features bearing maximum relevance to the class (C), it is highly possible that a fraction of the selected features is redundant i.e., a fraction of the features depend on one or more features selected. In such a situation, it is essential to adopt a condition to minimize this redundancy. Hence, the following Min-Redundancy condition is added to select mutually exclusive features:

$$min\, R(S),\; R = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i, x_j) \tag{3}$$

In order to achieve minimal-redundancy maximal-relevance (mRMR) there were two alternate (but not entirely equivalent) techniques proposed to combine the above two criteria:

$$max\, \phi_1(D, R),\; \phi_1 = D - R \tag{4}$$

$$max\, \phi_2(D, R),\; \phi_2 = D/R \tag{5}$$

The above two optimizations can be computed in O($|S|.M$) time. In this paper, technique 5 has been used.

3.2 Relevant Parallel Computing Considerations

Once an algorithm has been designed with optimal time complexity, any further speed-up requires the aid of hardware. In order to maximize this additional gain, it is essential that the algorithm is parallelized correctly.

There are two parallel models: data parallel model and task parallel model. In the former model, tasks are statically or semi-statically mapped onto processors. Each task performs similar operations on different data [19]. The latter model is typically employed to solve problems in which the amount of data associated with the tasks is large relative to the amount of computation associated with them [18]. In this paper we deal with the utilization of the Single Program Multiple Data paradigm.

Determining how the data is going to be split into chunks (group of contiguous iterations that are assigned to threads) is one of the most crucial steps while parallelizing an algorithm. There are multiple options for this step : static, dynamic and guided [20]. Static scheduling involves assigning a fixed number of chunks to each thread. This ensure lesser overheads but poor load balancing. Dynamic scheduling involves each thread grabbing iterations until all iterations are done. Consequently, faster threads are assigned more iterations. This ensures better load-balancing but incurs larger overheads than static scheduling. The last option is guided - a variant of dynamic scheduling where successive chunks get smaller. It is the best load-balancing option among the three techniques discussed but bears the highest overheads if the number of context switches are more and if the amount of interprocess communication demanded by the tasks is high [20]. The process of feature selection using mRMR involves less context switches and interprocess communication. Once threads have been allotted their task and have been assigned their relevant data chunks, they can carry out their task independently, without having to acquire information from other threads running in parallel. Hence, focus solely lies on load balancing, rendering guided as the preferred data scheduling technique.

The second factor that has to be kept in mind is the granularity. Depending on the amount of work which is performed by a parallel task, parallelism can be classified into three categories: coarse-grained, medium-grained and fine-grained parallelism. In coarse-grained parallelism, a program is split into small number of large tasks [21], leading to load imbalance.

In medium-grained parallelism, a program is split into smaller tasks thereby increasing the total number of tasks. It provides middle-ground between fine-grained and coarse-grained parallelism, where task sizes and communication times are greater than that of fine-grained parallelism but lesser than coarse-grained parallelism [21]. Lastly, we have fine-grained parallelism. In this a program is broken down to a large number of small tasks, facilitating load balancing. Inter process communication, however minute, adds up and greatly impedes the performance [21]. Reduction operations(for example, summation) take significant time as the results from multiple processes need to be aggregated. While hard-grained parallelism offers the lowest quality of load-balancing, fine-grained parallelism loses its advantage on account of its excessive sensitivity to the presence of even meager amount of inter process communication, which is inevitable for the case of parallel mRMR. Hence, the best alternative is to choose a medium-grained parallel technique.

3.3 Data Parallel mRMR

To implement the SPMD version of the mRMR algorithm, first we preprocess the mutual information values of the features with the class label. We then incrementally select features based on the maximum relevance and minimum redundancy criteria ensuring that the calculation of the relevance values happen in parallel. We also make use of an accumulator to ensure that the superfluous calculation of the relevances and redundancies is avoided.

Assume that the dataset X is a $n \times (M{+}1)$ matrix, having 'n' samples, 'M' columns representing the features and 1 column representing the class C. Let 'm' denote the number of features that have to be selected. Let us denote x_k as the vector corresponding to the *k-th* feature. Let L_c^i denote the candidate set of feature indices at step i and L_s^i denote the selected set of feature indices up until step i. Let R^- and R^+ be accumulators to store the total redundancy and total relevance, respectively, of the selected features. Let $R_j^- \, and \, R_j^+$ denote the redundancy and relevance, respectively, of the set of selected features if feature j is added to the list of selected feature indices at a given step. Let $I_{x_i,C}$ represent the mutual information between feature i and class C. Let R_{x_i,x_j} represent the mutual information between features x_i and x_j. Let us define $I(x,y)$ as a function that accepts two vectors as its parameters and returns as output their mutual information.

Below is the pseudo code for implementing the data parallel version of the mRMR algorithm. Note that the '**do in parallel**' phrase refers to executing the underlying loop segment across multiple threads with guided scheduling technique. Preferably, the number of threads is set to twice the number of cores.

A key observation that can be made after observing the initial algorithm[1] is that the steps corresponding to calculation of relevance measures and redundancy values for the candidate features are repetitive. Repetitive calculation of the relevance measures for each candidate feature can be optimized by precomputing the relevance value of each feature which is exactly what lines 3-5 in Algorithm 1 aim to achieve.

Algorithm 1: Data Parallel mRMR
Input: Dataset with M features, target class C
Output: 'm' features having minimum redundancy, maximum relevance
Pseudocode:

```
1: $L_c^1 = \{1, 2, ..., M\}$
2: $L_s^1 = \phi$
3: for i ←1 to M do in parallel
4:     $I_{x_i,C}$ = I($x_i$,C)
5: end for
6: $k^* = \text{argmax}(I_{x_i,C})$
7: $L_c^2 \leftarrow L_c^1 \setminus k^*$
8: $L_s^2 \leftarrow L_s^1 \cup k^*$
9: for i ← 1 to M do in parallel
10:     $R_{x_i,x_{k^*}} = R_{x_{k^*},x_i} = I(x_i, x_{k^*})$
11: end for
12: $R^- = 0$
13: $R^+ = I_{x_{k^*},C}$
14: for i ← 2 to m do
15:     for j ∈ $L_c^i$ do in parallel
16:         $R_j^- = R^- \times (i-1) \times (i-1)$
17:         for k ∈ $L_s^i$ do
18:             $R_j^- \leftarrow R_j^- + R_{x_j,x_k}$
19:         end for
20:         $R_j^- \leftarrow R_j^-/i^2$
21:         $R_j^+ \leftarrow ((R^+ \times (i-1)) + I_{x_j,C})/i$
22:         $mRMR_j = R_j^+/R_j^-$
23:     end for
24:     $k^* = argmax(mRMR_j)$
25:     for l ∈ $L_c^i$ do in parallel
26:         $R_{x_l,x_{k^*}} = R_{x_{k^*},x_l} = I(x_l, x_{k^*})$
27:     end for
28:     $R^- \leftarrow R_{k^*}^-$
29:     $R^+ \leftarrow R_{k^*}^+$
30:     $L_c^{i+1} \leftarrow L_c^i \backslash k^*$
31:     $L_s^{i+1} \leftarrow L_s^i \cup k^*$
32: end for
33: output $L_s^{m+1}$
```

Repetitive calculation of the redundancy values at each step for every candidate feature can be optimized by using a greedy approach as demonstrated by Lines 16-22 in Algorithm 1. Lines 25-27 in Algorithm 1 are for computing the mutual information values of each candidate feature with the most recently selected feature. The above two steps in conjunction with the use of the accumulator R^-, help in minimizing repetitive redundancy calculations and reducing the complexity from quadratic to linear.

4 Results

The scalability of data parallel mRMR was studied with the following specifications: Intel Core i5-6200U 2.4GHz processor, 2 cores and 16GB RAM. The scalability evaluation was carried out in two phases – first, the scalability on the number of samples was evaluated, followed by scalability on the number of features.

4.1 Scalability on number of samples

For the purpose of this experiment, the number of features was kept fixed at $1,000$ for 5 datasets. The number of samples was doubled for each dataset, starting from $6,400$. For each dataset, 200 features were to be selected. Observations made for varying number of threads are summarized in Table 1. ('N' is the number of samples, T_i denotes the time taken to select 200 features with 'i' threads, measured in seconds)

N	**T_1(s)**	**T_2(s)**	**T_3(s)**	**T_4(s)**
6,400	5.65	3.83	3.25	3.02
12,800	9.31	5.77	4.52	4.13
25,600	16.91	9.75	7.41	6.52
51,200	31.52	17.41	12.86	10.85
102,400	61.22	33.25	23.87	20.53

Table 1. Time scalability on the number of samples

Speedup (S) is defined in terms of sequential execution time ($T_{sequential}$) and parallel execution time ($T_{parallel}$) as:

$$S = \frac{T_{sequential}}{T_{parallel}} \tag{6}$$

Speedup is a commonly used metric in performance evaluation of parallel computing applications and discussions. To better understand the scalability on the number of samples, Table 2 presents the speedups obtained for varying number of samples across various number of threads ('N' is the number of samples, S_i denotes the speedup with 'i' threads).

To visualize the results better, the time taken for datasets with varying sizes was graphed against the number of threads (Fig. 1). It is evident from Fig. 1 that

N	$\mathbf{S_1}$	$\mathbf{S_2}$	$\mathbf{S_3}$	$\mathbf{S_4}$
6,400	1.0	1.47	1.73	1.87
12,800	1.0	1.61	2.05	2.25
25,600	1.0	1.73	2.28	2.59
51,200	1.0	1.81	2.45	2.90
102,400	1.0	1.84	2.56	2.98

Table 2. Speedup scalability on the number of samples

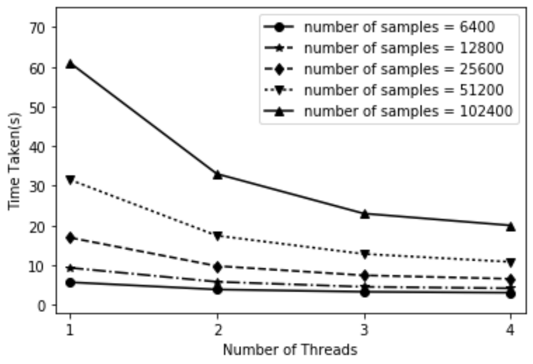

Fig. 1. Performance scalability for increasing sample size

with increasing number of threads, the time taken for feature selection decreases. This effect is more pronounced for larger datasets, which is also supported by the speedup values shown in Table 2.

4.2 Scalability on the number of features

For the purpose of this experiment, the number of samples was kept fixed at $5,000$ for 5 datasets. The number of features was doubled for each dataset, starting from $1,024$. For each dataset, 500 features were to be selected. Observations made for varying number of threads are summarized in Table 3 ('M' denotes the number of features, T_i denotes the time taken to select 500 features with 'i' threads, measured in seconds).

M	$\mathbf{T_1(s)}$	$\mathbf{T_2(s)}$	$\mathbf{T_3(s)}$	$\mathbf{T_4(s)}$
1,024	1.71	1.11	0.92	0.87
2,048	8.09	5.65	4.92	4.64
4,096	22.62	16.30	14.39	13.61
8,192	47.11	34.13	30.42	27.98
16,384	99.32	69.33	60.58	55.11

Table 3. Time scalability on the number of features

Proceeding in the same way as before, we observe the speedup obtained for varying number of features across varying number of threads shown in Table 4 (‘M’ denotes the number of samples, S_i denotes the speedup with ‘i’ threads).

M	$\mathbf{S_1}$	$\mathbf{S_2}$	$\mathbf{S_3}$	$\mathbf{S_4}$
1,024	1.0	1.55	1.85	1.96
2,048	1.0	1.43	1.65	1.74
4,096	1.0	1.38	1.57	1.66
8,192	1.0	1.38	1.54	1.68
16,384	1.0	1.43	1.63	1.80

Table 4. Speedup scalability on the number of features

To observe the time trend of feature selection for datasets of varying dimensionality across multiple threads a graph is shown in Fig. 2.

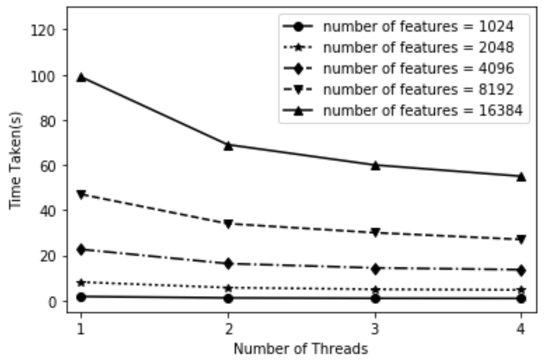

Fig. 2. Performance scalability for increasing number of features

From Fig. 2 and Table 4, it can be inferred that with increase in the number of threads, the time taken for feature selection decreases. However, as the dimensionality of the dataset increases, the total speed up achieved first decreases and then increases. This is in contrast with the case of increasing sample size, where the speedup continuously increases.

5 Conclusions and Future Work

In this paper, we proposed a novel data parallel approach to efficiently implement an improved version of the mRMR feature selection technique. Our data parallel approach provides load balancing, synchronous computation and higher speedup compared to existing task parallel approaches. Our algorithm effectively used preprocessing to reduce the time complexity of mRMR based techniques from quadratic to linear. The scalability of the proposed technique was studied across varying number of samples while keeping the dimensionality constant and also across varying dimensionalities while keeping the number of samples constant. For each case, the scalability was recorded in terms of speedup and time. It was observed that with increasing number of threads the speedup increased and this effect was more pronounced for larger number of samples. However, as the dimensionality increases, the speedup first decreases and then increases.

The current approach used an auxiliary space of $O(m^2)$ where 'm' represents the number of features. In the future, we aim at developing an advanced approach to lower the space complexity and also to effectively incorporate the proposed algorithm into embedded feature selection techniques.

References

1. H. Peng, F. Long and C. Ding, "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy", in IEEE Transaction on Pattern Analysis and Machine Intelligence, 27, pp. 1226-1238, 2005
2. D. Lu and Q. Weng, A survey of image classification methods and techniques for improving classification performance, in Journal of Remote Sensing, 28, pp.823-870, 2007.
3. A. Jain and D. Zongker, Feature selection: evaluation, application, and small sample performance, in IEEE Transaction on Pattern Analysis and Machine Intelligence, 19, pp.153-158, 1997
4. C. Bhattacharyya et al., "Simultaneous relevant feature identification and classification in high-dimensional spaces: Application to molecular profiling data." Special issue on Genomic Signal Processing, Vol. 83, No.4, pp. 729-743, 2003.
5. A.A. Alizadeh et al. Distinct Types of Diffuse Large B-Cell Lymphoma Identified by Gene Expression Profiling, Nature, vol. 403, pp. 503-511, 2000.
6. Thomas, J.G. et al. An efficient and robust statistical modeling approach to discover differentially expressed genes using genomic expression profiles, Genome Research, 11, 1227-1236, 2001
7. R. Kohavi and G. H. John, Wrappers for feature subset selection, Artif. Intell., vol. 97, no. 1-2, pp. 273-324, Dec. 1997
8. P. Narendra, K. Fukunaga A branch and bound algorithm for feature subset selection IEEE Trans Comput, 6 (1977), pp. 917-922
9. R. Kohavi, G.H. John Wrappers for feature subset selection Artif Intell, 97 (1997), pp. 273-324
10. M. L. Raymer, W.F. Punch and E.D. Goodman, Dimensionality reduction using genetic algorithms, in IEEE Transactions on Evolutionary Computation, 4, pp. 164-171, 2000
11. Kennedy J, Eberhart RC. Particle swarm optimization. In: Proc IEEE intl conf on neural networks, IV; 1995. p. 1942-1948.
12. D. Ververidis and C. Kotropoulos, "Sequential forward feature selection with low computational cost," 2005 13th European Signal Processing Conference, Antalya, 2005, pp. 1-4.
13. LeKhac N., Wu B., Chen C., Kechadi MT. (2013) Feature Selection Parallel Technique for Remotely Sensed Imagery Classification. In: Murgante B. et al. (eds)

Computational Science and Its Applications – ICCSA 2013. ICCSA 2013. Lecture Notes in Computer Science, vol 7972. Springer, Berlin, Heidelberg

14. J. T. de Souza, S. Matwin, and N. Japkowicz, Parallelizing feature selection, Algorithmica, vol. 45, no. 3, pp. 433456, 2006.
15. C. Ding, H. Peng, "Minimum Redundancy Feature Selection from Microarray Gene Expression Data", in Journal of Bioinformatics and Computational Biology, pp. 523-528, vol.3, 2003.
16. S. Ramrez-Gallego et al. An information theory-based feature selection framework for big data under apache spark, IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. PP, no. 99, pp. 113, 2017.
17. Ramrez-Gallego, Sergio et al. Fast-mRMR: Fast Minimum Redundancy Maximum Relevance Algorithm for High-Dimensional Big Data. Int. J. Intell. Syst. 32 (2017): 134-152.
18. Reggiani, Claudio et al. Feature selection in high-dimensional dataset using MapReduce., BNCAI (2017).
19. N-A. Le-Khac, "Studying the performance of overlapping communication and computation by active message: Inuktitut case", International Conference on Parallel and Distributed Computing and Network (PDCN'06), February 12-14, 2006, Innsbruck, Austria
20. Ayguad E. et al. (2003) Is the Schedule Clause Really Necessary in OpenMP?. In: Voss M.J. (eds) OpenMP Shared Memory Parallel Programming. WOMPAT 2003. Lecture Notes in Computer Science, vol 2716. Springer, Berlin, Heidelberg
21. Tick, E. NGCO (1990) 7: 325. https://doi.org/10.1007/BF03037210
22. Alshamlan H, Badr G, Alohali Y. mRMR-ABC: A Hybrid Gene Selection Algorithm for Cancer Classification Using Microarray Gene Expression Profiling. Biomed Res Int. 2015;2015:604910.
23. Enireddy, Vamsidhar, Dr DVVS PhaniKumar, and Dr Gunna Kishore. "Application of Fisher Score and mRMR Techniques for Feature Selection in Compressed Medical Images." International Journal of Engineering and Technology (IJET) 7.6 (2016).
24. Kaya, Heysem, et al. "Random forests for laughter detection." Proceedings of Workshop on Affective Social Speech Signals-in conjunction with the INTERSPEECH. 2013.
25. Alomari, OSAMA AHMAD, et al. "Mrmr Ba: a hybrid gene selection algorithm for cancer classification." J Theor Appl Inf Technol 95.12 (2017): 1.
26. Li, Zhao, et al. "A parallel feature selection method study for text classification." Neural Computing and Applications 28.1 (2017): 513-524.
27. Zhou, Y., Porwal, U., Zhang, C., Ngo, H. Q., Nguyen, X., R, C., & Govindaraju, V. (2014). Parallel feature selection inspired by group testing. In Advances in Neural Information Processing Systems (pp. 3554-3562).