2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015

Performance Evaluation of Statistical Functions

André Rodrigues*, Carla Silva*, Paulo Borges†, Sérgio Silva†, Inês Dutra‡
*INESC TEC, Portugal
†NLPC Lda., Portugal
‡CRACS INESC TEC and University of Porto, Portugal

***Abstract*—Statistical data analysis methods are well known for their difficulty in handling large number of instances or large number of parameters. This is most noticeable in the presence of "big data", i.e., of data that are heterogeneous, and come from several sources, which makes their volume increase very rapidly. In this paper, we study popular and well-known statistical functions generally applied to data analysis, and assess their performance using our own implementation (DataIP)[1], MatLab and R. We show that DataIP outperforms MatLab and R by several orders of magnitude and that the design and implementation of these functions need to be rethought to adapt to today's data challenges.**

***Keywords*-data analysis; statistical functions; performance evaluation; MatLab; R;**

I. Introduction

As the amount of information grows more than exponentially along the years, we start to face a big data challenge. In order to process and analyze all these data, we need new tools and algorithms that can cope with their heterogeneity and volume. Statistical data analysis methods are well known for their difficulty in handling large number of instances or large number of parameters. Likewise, machine learning algorithms may require large amounts of running-time. These problems are most noticeable in the presence of "big data", i.e., of data that are heterogeneous, and come from several sources, which makes their volume increase very rapidly.

Often just accessing the data may be a serious bottleneck. These considerations motivated work on the so-called "noSQL" data-bases, such as Google's BigTable [1], or the Apache Foundation's Spark [2]. The latter supports distributed data, and includes MLLib [3], an implementation of several machine learning algorithms. In contrast to transaction oriented data-bases, statistical methods are often applied to data repositories with read-only data, which avoids data consistency overheads. Progress in DRAM technology has enabled manipulating large amounts of data in main memory. However, these solutions require specialized hardware and software to be able to handle massive amounts of data. In general, in order to analyze data, most people use traditional tools such as Excel, SPSS, MatLab or R. Although these are very popular, they are not tailored to handle large amounts of data. In this work, we show that we can process and analyze millions of instances, with potentially hundreds of variables, faster than these systems, with the same simplicity, and without the need of specialized resources, except the use of top-of-the-shelf multicore machines.

We concentrate on the following bivariate and multivariate analysis, which are most popular in data analysis: Chi-square and Shapiro-Wilk tests, Pearson, Spearman and Kendall correlations, linear and logarithmic regression, Wilcoxon, Kolmogorov-Smirnov (K-S), T-student and Bartlett statistical tests. We also study the performance of calculating data summaries. Our results are always superior to MatLab and R, and our best result, for Chi-square, achieves a performance 75% better than R.

Our software is written in C/C++, and implements the same functions available in MatLab and R, producing the same results for the same inputs. Next, we describe the main functions our software implements, discuss about implementation issues, present our experimental methodology and results, and finally, we conclude.

II. Background and Methodology

In a data mining process, an important issue is the pre-processing phase. Data preparation and selection is relevant for knowledge discovery. Next, we have the data analysis phase, where usually we start to "know" the data using descriptive and inferential statistics through univariate, bivariate and multivariate methods.

The major purpose of univariate analysis is to describe the data. Univariate analysis is usually used in the first descriptive stages of problem solving, being complemented by more advanced, inferential bivariate or multivariate analysis. Descriptive statistics [4] describe and summarize data. Univariate descriptive statistics describe individual variables. Exploratory analysis of data gives us a summary statistics of measures of Central Tendency, Dispersion and Shape of the data.

Measures of central tendency locate a distribution of data along an appropriate scale, such as: geometric mean, harmonic mean, arithmetic average, median and mode. The purpose of measures of dispersion is to find out how spread out the data values are. Another term for these statistics is measures of spread: interquartile range, percentiles, average absolute deviation (or simply called average deviation),

[1]DataIP is a R&D project from NLPC - Dataias. All contacts related to this work should be addressed to: rd@dataias.com, http://www.dataias.com

978-1-5090-1893-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SmartCity.2015.159

range, standard deviation and coefficient of variation. This analysis is usually followed by measures of shape. The measures of shape indicate the symmetry and flatness of the distribution of a data sample. A distribution of data item values may be symmetrical or asymmetrical. In this field we have: variance, kurtosis, central moment of 3rd order and Pearson asymmetry coefficients G1 and G2.

Bivariate analyses are conducted to determine whether a statistical association exists between two variables, the degree of association if one does exist, and whether one variable may be predicted from another. It deals with causes or relationships. The major purpose of bivariate analysis is to: (1) Define the nature of the relationship, (2) Identify the type and direction of the relationship, (3) Determine if the relationship is statistically significant, and (4) Identify the strength of the relationship.

In bivariate analysis we use nonparametric statistics [5], given that our focus is on inferential statistics [6]. We use the same hypotheses testing used in the univariate analysis, as well as other tests specific to bivariate analysis (Kolmogorov-Smirnov, Wilcoxon and T-test).

Multivariate studies are analogous to bivariate studies, but involve multiple variables. Researchers could then use multivariate statistical analysis to study the relationships between all of the variables. Multivariate analytical techniques symbolize a variety of mathematical models used to measure and quantify outcomes, taking into account relevant factors that can cause this relationship. The most common is multiple regression analysis [7] whose objective is to understand how the value of the dependent variable changes when any of the independent variables is varied, while the other independent variables are fixed, using multiple response variables. With regression [8] we attempt to find a function which models the data with the minimum error.

III. Implementation

Based on the whole data analysis process, we developed a framework that can make use of tools for univariate, bivariate, multivariate statistical analysis and also include some machine learning methods. In this paper, we concentrate on the statistical methods. In a general context, knowledge extraction is performed through exploratory analysis, univariate, bivariate and multivariate, and using descriptive and inferential statistics. We have optimized sequential and parallel implementations of the methods. Moreover, we take advantage of the fact that several statistical functions share common data preprocessing operations, and execute them only once to speedup execution. For example, correlation functions usually sort the two variable values to be correlated. We sort all variables beforehand, store the resulting vectors, and reuse then whenever needed. We also store minimum and maximum values, modes and medians, among others.

The algorithms were developed in C/C++ and implement well known functions used in R and MatLab.

We implemented all of those methods from scratch, in C/C++, using the best algorithms found in the literature, optimizing for performance, and removing redundant calculations. For example, as we have all functions integrated, lots of calculations can be saved because they are common to several methods. One example is sorting variable values, which is performed just once. Memory usage is other important issue, and we needed to organize our implementation to minimize writes and memory allocation during calculations (reads are always faster than writes). Another issue on the implementation is the complexity of the algorithms. Our implementations keep a maximum complexity of $O(nlogn)$.

In order to get the most possible organized structure and layer separation, we are using an object oriented structure in C++.

Our most important optimization is on the calculation of Kendall τ (correlations). This is known to have quadratic complexity, but we use Knight's algorithm [9], [10], that calculates Kendall τ with $O(nlogn)$ complexity.

IV. Parallelization

Our parallel implementation runs on multicore machines. We use openMP [11] for parallelization. In order to get big chunks of processing and always get speedups avoiding overheads, parallelism is used on the following steps:

- On calculating data summary, all statistics related to this procedure are calculated in parallel, attribute by attribute. If we have only one attribute, the calculations are sequential, but if we have more than one, they are performed in parallel.
- On calculating correlations, all pairs are calculated in parallel. Each core processor calculates one pair with all three correlations (Pearson, Spearman and Kendall).
- On multivariate analysis the idea is the same. Each core calculates all analysis to each one of the permutations.

V. Experimental methodology

We performed our experiments in three different machines: a plain multicore workstation with 4 cores (hyperthreaded allowing 8 threads to run simultaneously), 16 GBytes of memory and Linux Fedora 18 (**Machine 1**); a Xeon server with 8 cores (16 threads), 24 GBytes of memory and Fedora 20 (**Machine 2**); an IBM system x3755 with 6 cores, 48 GBytes of memory; and Windows Server 2012 Datacenter (**Machine 3**).

In order to apply the statistical analysis, we used two datasets: real data collected from Hospital das Clínicas (São Paulo, Brazil), consisting of around 200,000 patient discharges (**Dataset 1**) with 26 variables, and a synthetic dataset created with millions of instances, but with only two variables because of memory constraints (this dataset alone is almost 5 GigaBytes large) (**Dataset 2**).

We use Dataset 1, patient discharges, with different versions:

- 300 version: It is a subset with 300 rows, and 9 numeric variables without nulls.
- 200k version: original dataset consisting of 201,879 discharge patients, and 9 numeric variables without nulls
- 1M version: it is a replication in chunks of the original dataset 200k in order to achieve one million instances, with 6 numeric variables without nulls.
- 5M version: it is a replication in chunks of the original dataset 200k, in order to achieve five million instances, with 6 numeric variables without nulls.
- 10M version: it is a replication in chunks of the original dataset 200k, in order to achieve ten million rows, with 6 numeric variables without nulls.

Dataset 2 is a synthetic dataset created to study the scalability of our implementation when handling many millions of instances. It has only two variables, due to the memory limits of our machines. The dataset was generated with random numbers between 0 and 1235. The type of data and range do not affect results. The file size is 4.8 GBytes.

Most of our experiments were run with Dataset1 using Machine1. Dataset2 ran in the last two machines (the dual Xeon and the IBM servers, Machine2 and Machine3, respectively). The experiment in Machine2 compares the Kendall results running on Linux and Windows with only one thread and a larger set of instances, as in Dataset2. The experiment in Machine3 compares the execution of Kendall on Oracle with our implementation. Kendall is highlighted here because we use an algorithm whose complexity is $O(nlogn)$ as compared with the quadratic implementations of MatLab, R or Oracle.

We performed experiments for the summary, Chi-square and Shapiro-Wilk tests, the three correlations (Pearson, Spearman and Kendall), linear and logarithmic regression, Wilcoxon, K-S, T and Bartlett statistical tests. Some experiments were aborted by MatLab or R, because of lack of memory or because they exceeded our maximum running time limit.

All experiments in Machine1 and Dataset1 were run with 1 and 4 threads. In our implementation, we measured execution times necessary to execute each one of the statistical functions, but because some operations need to be computed only once (for example, sorting variable values), this time is computed only once. A list of functions timed in MatLab is: `corr(var1,var2,'type','Pearson')`, `corr(var1,var2,'type','Spearman')`, `corr(var1,var2,'type','Kendall')`, `fitlm`, `min`, `max`, `median`, `mode`, `mean`, `geomean`, `harmmean`, `std`, `var`, `range`, `iqr`, `mad`, `quantile`, `prctile`, `kurtosis`, `moment`, `ranksum`, `kstest2`, `ttest2`. A list of functions timed in R is: `chisq.test`, `shapiro.test`, `cor(var1,var2,method="pearson")`, `cor(var1,var2,method="spearman")`, `cor(var1,var2,method="kendall")`, `lm(var1 var2)`, `summary(lm(var1 var2))`, `min`, `quantile`, `median`, `mean`, `max`, `var`, `sd`, `IQR`, `range`, `kurtosis`, `wilcox.test`, `ks.test`, `t.test`, `bartlett.test`.

Time to read the input data was not taken into account in any of the experiments.

Shapiro-Wilk was not run for all dataset sizes, because the algorithm only allows input sizes of at most 5000 rows [12].

Linear and logarithmic regression, and the Wilcoxon, Kolmogorov-Smirnov (K-S), T-student and Bartlett statistical tests are only applied to datasets that have strong correlations between variables (correlation value greater than 0.75 with confidence greater than 95%). As this is true only for the 300 dataset, results are shown just for this dataset for these functions.

VI. Experiments and Results

Tables I to IX show execution times, in seconds, for each statistical function, using our implementation (DataIP (1) – with a single thread, and DataIP (4) – with 4 threads), MatLab and R, running on Machine1, applied to datasets 300, 200k, 1M, 5M and 10M.

A. Summary

Table I shows execution times for MatLab, R and our implementation of the summary for 1 thread (DataIP (1)) and for 4 threads (DataIP (4)), with varying dataset sizes. All experiments were run on Machine1. For this task, R is twice as fast as MatLab. DataIP (1) is almost ninety times faster than R, and almost two hundred times faster than MatLab. If we look at the parallel implementation of DataIP, which uses 4 threads, the speedup is almost double.

Table I
Summary (time in seconds).

	300	**200k**	**1M**	**5M**	**10M**
MatLab	0.109624	1.215	2.733	73.835	-
R	0.05	0.529	2.778	19.716	46.713
DataIP (1)	0.000569	0.298	1.583	8.31	16.063
DataIP (4)	0.000319	0.153	0.814	4.01	8.32

DataIP with one thread (DataIP (1)) takes 0.000569 seconds to compute the summary for the 300 dataset while the 4 threads version takes 0.000319 seconds, being 43.9% faster than the sequential version. As we increase the dataset size, this gain is maintained with DataIP with 4 threads keeping an average speedup of 2 over the 1-thread version. Notice that our parallelization is very simple and only computes in parallel the summary of each attribute. As these datasets have a small number of non-null attributes, we do not take full advantage of the multi-threaded version, but still manage

to have an efficiency of almost 50% for all dataset sizes, doubling the speed of the sequential version.

We do not have access to a parallelized MatLab toolbox, but, even if the 4-threads version speedup of MatLab was perfect, DataIP with 4 threads would still be almost 86 times faster than MatLab with 4 threads to calculate the summary for the 300 dataset. The same is true for the other dataset sizes. If R were run in parallel with 4 threads and would have perfect speedup, it would be 40 times slower than DataIP (4).

Figure 1 shows how the three implementations compare in terms of execution times, in seconds, as we vary the dataset sizes.

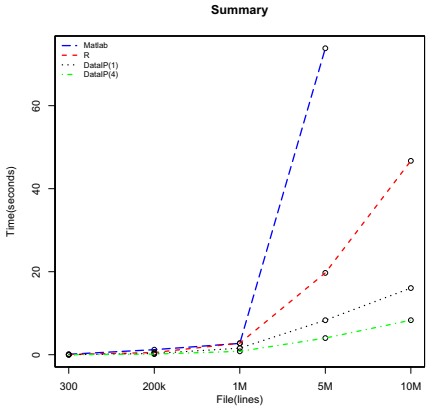

Figure 1. Summary Execution times (seconds).

B. Chi-square test

We ran the chi-square test only in R, because the `chi2gof` function of MatLab gives a result that is very much different than R or our implementation. It looks like it does something else more than just performing the chi-square test.

Table II
CHI-SQUARE TEST (TIME IN SECONDS).

	300	200k	1M	5M	10M
R	0.025	1.479	3.076	8.351	15.106
DataIP (1)	0.000250	0.259	1.285	6.288	12.597
DataIP (4)	0.000089	0.063	0.325	1.612	3.214

For this function we got better performances than for the summary. Speedups are also higher. DataIP (4) has speedup 2.8 on 4 threads when compared with DataIP running with a single thread, for the 300 dataset. When comparing with R, DataIP (1) is 100 times faster and DataIP (4) is almost 281 times faster, for the 300 dataset.

Figure 2 shows how the three implementations compare in terms of execution times, in seconds, as we vary the dataset sizes.

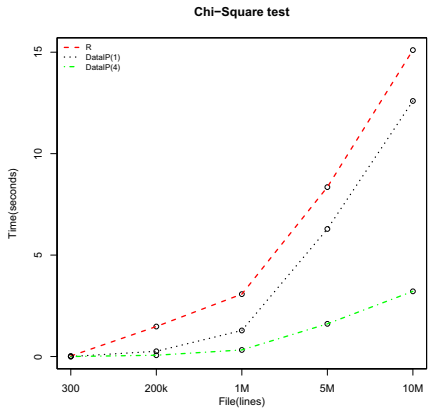

Figure 2. Chi-Square test execution times (seconds).

C. Shapiro-Wilk test

Table III
SHAPIRO-WILK TEST (TIME IN SECONDS).

	300
R	0.073
DataIP (1)	0.000176
DataIP (4)	0.000147

The Shapiro-Wilk test performed again much better in DataIP than in R (more than 400 times faster), but with a modest speedup from 1 to 4 threads. This function is not directly implemented in MatLab, that is why we do not show results for MatLab.

D. Pearson Correlation

Table IV
PEARSON CORRELATION (TIME IN SECONDS).

	300	200k	1M	5M	10M
MatLab	0.428668	0.582974	1.250868	24.112243	(aborted)
R	0.011	1.259	14.045	8.07	15.741
DataIP (1)	0.000026	0.003780	0.022880	0.115164	0.238069
DataIP (4)	0.000025	0.002709	0.018669	0.090914	0.197789

For the Pearson correlation we did not reach speedups as good as the other statistical functions, running only 3.9% faster for the 300 dataset, with just a drop of 1 microsecond. For the 200k it achieves its best results (28.3% faster), followed by 1M (18.4% faster), 5M (21.1% faster) and finally 10M (16.9% faster).

E. Spearman Correlation

For the Spearman correlation we got the best speedup for the 300 dataset (DataIP (1) is 2.3 times faster than DataIP (4)). Speedups for the larger datasets are limited, but in

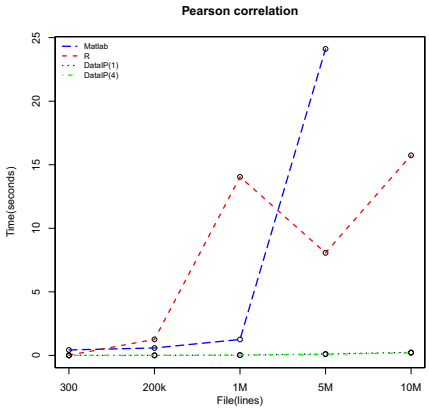

Figure 3. Pearson Correlation.

Table V
SPEARMAN CORRELATION (TIME IN SECONDS).

	300	200k	1M	5M	10M
MatLab	0.147789	3.428456	37.366954	41.568466	(aborted)
R	0.024	6.115	78.812	206.6	477.634
DataIP (1)	0.000413	0.048796	0.471813	2.792287	5.567729
DataIP (4)	0.000174	0.026701	0.277256	1.589052	3.276365

any case, the DataIP implementation largely outperforms MatLab and R. The 10M dataset could not run on MatLab. Figure 4 shows execution times as we increase the dataset sizes.

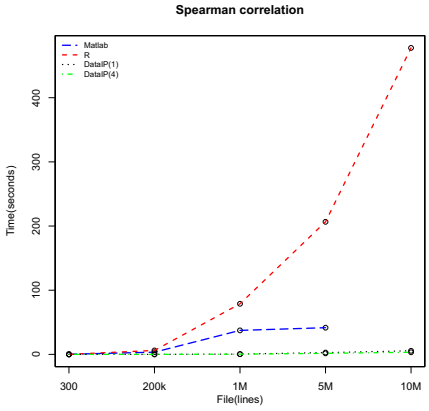

Figure 4. Spearman Correlation.

F. Kendall Correlation

Regarding the Kendall correlation we got an overall good speedup performance. For all dataset sizes we managed to have an average of 2.5 speedup with 4 threads related to the single threaded version. Once more, we can notice the long

Table VI
KENDALL CORRELATION (TIME IN SECONDS).

	300	200k	1M	5M	10M
MatLab	0.453558	17214.175	13809.805	(aborted)	(aborted)
R	0.212	37927.22	(aborted)	(aborted)	(aborted)
DataIP (1)	0.000535	0.202866	1.09301	6.625	14.143
DataIP (4)	0.000480	0.075935	0.428808	2.616	5.687

processing times for MatLab and R. For the 200k, MatLab took 4.8 hours and for the 1M, 3.8 hours. R took about 10.5 hours for the 200k, the worst case scenario we got in our tests. Implementations of Kendall usually employ the quadratic algorithm, but we use an $O(nlogn)$ complexity version published many years ago [9]. This can partially explain the poor performance achieved by MatLab and R.

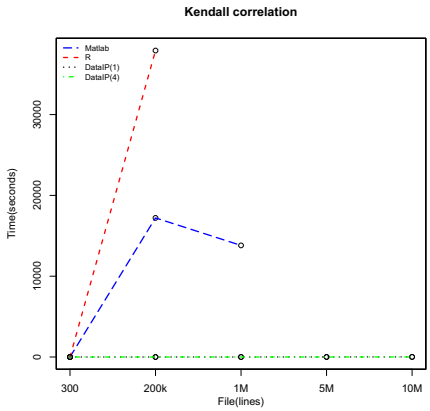

Figure 5. Kendall Correlation.

G. Linear and Logarithmic regression

Table VII
LINEAR AND LOGARITHMIC REGRESSION (TIME IN SECONDS).

	300
MatLab	1.209489
R	0.032
DataIP (1)	0.000757
DataIP (4)	0.000449

For processing the two functions for linear and logarithmic regression, the speedup was 1.68 when running with 4 threads, for the 300 dataset. R runs the same regressions, producing the same results, being two orders of magnitude slower than DataIP.

H. Wilcoxon, K-S and T tests

Table VIII shows the execution times for the three statistical tests we implemented: Wilcoxon, Kolmogorov-Smirnov (K-S) and T tests. The speedup is modest and again

Table VIII
WILCOXON, K-S AND T TESTS (TIME IN SECONDS).

	300
MatLab	0.123508
R	0.03
DataIP (1)	0.000188
DataIP (4)	0.000141

DataIP outperforms R and MatLab by at least two orders of magnitude.

I. Bartlett test

Table IX
BARTLETT TEST (TIME IN SECONDS).

	300
R	0.071
DataIP (1)	0.000004
DataIP (4)	0.000003

We ran the Bartlett test only in R, because the Bartlett function of MatLab gives a result that is very much different than R or our implementation. It looks like it does something else more than just performing the test.

This test [13] ends our experiments and results on Machine1, for the patients discharge dataset. DataIP did not achieve any speedup, but performed three orders of magnitude faster than R.

J. Kendall Correlations Performance on Big Data

In this section, we present results for the comparison between the same software running on Linux and Windows, using a larger number of instances (Dataset 2), on Machine 2. The dataset has 300 million instances. The tests are run on a single thread, since the dataset has only two variables and we parallelize the column operations.

Table X
KENDALL CORRELATIONS WITH DATAIP (TIME IN SECONDS).

	300M
Windows	116.489
Linux	113.264

Linux is slightly faster than Windows. In both environments our implementation performs well running the Kendall correlation in less than 2 minutes. Unfortunately, we could not load the dataset in R or MatLab.

K. Kendall Correlations on Oracle Database 12c enterprise

In this section we compare DataIP with the Oracle implementation of Kendall, in Machine3, the Windows Server 2012 Datacenter using Dataset2.

The Oracle implementation has time complexity that is probably quadratic. In order to perform this experiment, we had to reduce our 300 million instances. We created two smaller subsets: one with about 50 thousand instances and another one with about 200 thousand instances.

Table XI
KENDALL CORRELATION (TIME IN SECONDS).

	50k	200k
Oracle 12c Enterprise	426.46	6253.568
DataIP (1)	0.015	0.046

Table XI shows the results for this experiment. To get exactly the same results, on the 50k subset, DataIP is 28430.7 times faster than Oracle 12c Enterprise, and on the 200k subset is 135947.1 times faster than Oracle 12c Enterprise. With larger datasets, we would even better results, since on the Oracle 12c Enterprise the execution time grows faster than on DataIP.

VII. CONCLUSION AND FUTURE WORK

Our main conclusion of this work is that traditional implementations of basic statistical functions, crucial for data analysis, need to be revisited, and better designed to meet the requirements of larger datasets. We presented results of a C/C++ implementation of many statistical functions and show that, even for small datasets, well designed code can achieve very good performance when compared with state-of-the-art statistical software. Experiments running with 1 thread or 4 threads perform several orders of magnitude faster than R or MatLab. Besides, we can achieve reasonable speedups taking advantage of multicore, which MatLab and R can also take, but, even with perfect speedups, would not beat DataIP.

We would like to continue testing other functions, comparing our parallel implementation with parallel versions of the same functions in R and MatLab, and improving our parallel implementation, which, currently, achieves modest speedups.

ACKNOWLEDGEMENTS

We are grateful to Prof. Domingos Alves, from Faculty of Medicine of Ribeirão Preto, São Paulo, Brazil, who kindly provided us with the patient discharge dataset. This work was supported by NLPC, Lda (Proj. DATAIP Nb. 38667, 07/2012-SII&DT), IAPMEI/COMPETE/QREN/EUROPEAN UNION, Copyright ©2014 2015 NLPC – I.C.NLPC T.A.I.C.C.Gestão, LDA – All rights reserved. Other funding, which supported attendance to the DataCom conference, was provided by FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013.

REFERENCES

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26,

Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365815.1365816 1

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113 1

[3] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *CoRR*, vol. abs/1505.06807, 2015. [Online]. Available: http://arxiv.org/abs/1505.06807 1

[4] D. Rumsey, *Statistics Essentials For Dummies*, ser. – For dummies. Wiley, 2010. [Online]. Available: https://books.google.pt/books?id=QBmsVY0p7YkC 1

[5] W. Conover, *Practical Nonparametric Statistics*, ser. Cram101 Series. Cram101 Incorporated, 2006. [Online]. Available: https://books.google.pt/books?id=ouvPSgAACAAJ 2

[6] G. Casella and R. Berger, *Statistical Inference*, ser. Duxbury advanced series. Duxbury Thomson Learning, 2008. [Online]. Available: https://books.google.pt/books?id=ZpkPPwAACAAJ 2

[7] D. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 2010. [Online]. Available: https://books.google.pt/books?id=_f4KrEcNAfEC 2

[8] T. Wonnacott and R. Wonnacott, *Student workbook, Introductory statistics for business and economics, fourth edition and Introductory statistics, fifth edition*. Wiley, 1990. [Online]. Available: https://books.google.pt/books?id=HuUIAQAAMAAJ 2

[9] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," *Journal of the American Statistical Association*, vol. 61, no. 314, pp. pp. 436–439, 1966. [Online]. Available: http://www.jstor.org/stable/2282833 2, 5

[10] D. Christensen, "Fast algorithms for the calculation of kendalls ," *Computational Statistics*, vol. 20, no. 1, pp. 51–62, 2005. [Online]. Available: http://dx.doi.org/10.1007/BF02736122 2

[11] R. Chandra, *Parallel Programming in OpenMP*, ser. High performance computing. Morgan Kaufmann Publishers, 2001. [Online]. Available: https://books.google.pt/books?id=18CmnqIhbhUC 2

[12] P. Royston, "Remark as r94: A remark on algorithm as 181: The w-test for normality," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 44, no. 4, pp. 547–551, 1995. [Online]. Available: http://www.jstor.org/stable/2986146 3

[13] M. S. Bartlett, "Properties of sufficiency and statistical tests," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 160, no. 901, pp. 268–282, 1937. 6