Performance evaluation of various distance metrics in agglomerative clustering

Arnold Szederjesi
Department of Computer Science
Babeş-Bolyai University
1 M. Kogalniceanu Street, 400084, Cluj-Napoca, Romania
Email: saie1486@scs.ubbcluj.ro

***Abstract*—In hierarchical clustering algorithms the choice of the metric is essential, because it will influence the whole process, as some elements may be close related to one distance and far away according to another. There are many distances that are applied in our agent-based clustering algorithm and thus the paper presents an illustration of how this distances actually work. Abstractul este un rezumat al articolului. Unele reviste sunt chiar scumpe; la fiecare articol se poate vedea gratuit numai abstractul; acesta trebuie sa ma convinga sa cumpar revista/articolul. Abstractul il scriem ultimul.**

I. Introduction

In data mining, clustering is a very important aspect for data grouping. In many other fields like Biology[1], Bioinformatics[2], Chemistry[3] data classification is essential, but it is not trivial. The need of such kind of algorithms is high in these domains. So we implemented an agglomerative hierarchical clustering algorithm. Coming from the nature of these clustering approaches, the choice of the metric to be used when combining similar data is important but not that simple. Every dataset is different, some datasets can be separated linearly, some not[4]. This is why is important to adapt the metric used to the dataset on which is going to be applied. Considering this problem in this paper we apply several metrics to see how they perform in a real environment.

The paper is organized as follows: Section II describes our motivation to develop such an application, Section III presents related works that are based on the same datasets, Section IV consists of information needed to understand the concepts that follow, Section V shows and explains the proposed algorithm, and presents the distances used, Section VI reflects the ideas regarding the datasets, and how we applied the algorithm, Section VII illustrates the results with tables and some analysis on them, and finally Section VIII draws the conclusions of this paper and presents ideas for future work.

II. Motivation

The goal of this paper is to show how hierarchical clustering is viable even for large datasets, and to emphasize the importance of the chosen metric. We want to show that for different datasets different metrics have to be used to achieve the best results possible. The advantage of hierarchical clustering algorithms is the fact that the number of clusters does not need to be known in advance. On the other hand a big disadvantage of them is their computational expensiveness.

The idea to overcome this is to parallelize the clustering process. This is achieved by using a MAS (multi-agent system) [5] which consists of agents (one agent for each data). Multi-agent systems are suitable for concurrent execution of different tasks, the agents have to work together in order to achieve higher order goals, which could not be achieved by a single agent. In this case the use of agents is important for concurrent clustering of data, which makes even a computationally expensive algorithm to run as a linear one. These agents are implemented to run on processes provided by the Erlang [6] VM. Some important projects that run on the Erlang Virtual Machine are the chats from Facebook and Whatsapp. Also to avoid the negative experience, the boilerplates and the code duplications needed when programming in Erlang we rather used Elixir [7] [8]. Elixir is a functional, concurrent, general-purpose, meta-programming aware programming language that runs on the Erlang Virtual Machine, adopting the capabilities of building concurrent, distributed and fault-tolerant applications. The main difference between Elixir/Erlang and the other programming languages is that they provide concurrency automatically. As Jos Valim, the creator of Elixir says in [8]: ”As garbage collection once freed developers from the shackles of memory management, Elixir is here to free you from antiquated concurrency mechanisms...”. Also languages like Java, or C do not support meta-programming, which sometimes can make a programmer's life easier. Elixir opposed to Erlang, also supports polymorphism and first-class tooling, whereas also bringing joy when writing code.

III. Related work

aici ar trebui luate pe rand articolele altora si zici un pic ce/cum au facut. La fiecare articol sa fie macar 5-6 randuri.

On the datasets Iris [4] and Cancer [9] we choose the article [10]. In this paper they made an algorithm which works with distance approximation to build the clusters. For the dataset Seeds [11] the article [12] was chosen. The focus in this paper is on the algorithm they designed for clustering, and is compared with the K-means [13].

IV. Theoretical Background

In machine learning, clustering is an example of unsupervised learning because it does not rely on predefined classes and class-labelled training examples. So it could be said that clustering is a form of learning by observation, rather than learning by examples. In data analysis, efforts have been conducted on finding methods for efficient and effective cluster analysis in large databases. The main requirements for a good clustering algorithm would be the scalability of the method, its effectiveness for clustering complex shapes and types of data, dealing with high-dimensional data, and handling mixed numerical and categorical data in large databases [14].

Definition 4.1: Clustering is the process of finding a set

$$\mathcal{C} = \{C_k | k = \overline{1,p}\} \tag{1}$$

of subsets of a given set of objects

$$X = \{x^i | i = \overline{1,n}, 1 \leq p \leq n\} \tag{2}$$

such that:

- $\forall k, l = \overline{1,p}, k \neq l : C_k \cap C_l = \emptyset$
- $\bigcup_{k=1}^{n} C_k = X$
- $\forall k = \overline{1,p} : C_k \neq \emptyset$
- $\forall k = \overline{1,p}, \forall i, j = \overline{1, |C_k|} : sim(x^i, x^j)$ — holds,

where: $sim(x^i, x^j)$ denotes the similarity between two items x^i and x^j.

Hierarchical clustering [15], also known as hierarchical cluster analysis, is a technique which aims for building a hierarchy of clusters. There are two strategies to achieve this:

- agglomerative: initially each instance is in its own cluster, and they are merged until a reasonable number of clusters remain
- divisive: at the start all the observations are in one cluster, and this is splitted into smaller clusters

Multi-agent Systems are systems composed of agents that interact with each other to achieve higher goals, that a single agent could not be able to reach. As Shoham says: "Most often, when people use the term agent they refer to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist." [16] Agents should behave autonomously, parallel and based on stimulation provided by the environment they are set in. It is a computerized simulation of human behaviour in a company, in a class or even in the street. MAS can be used to solve difficult problems, that need concurrency and "teamwork". In our case is important to use MAS to make the clustering work in parallel to reduce the complexity of the algorithm. It is also good to remark that without agents this kind of approach would not be possible, because there is the need of continuous communication between data.

V. Proposed approach

We propose an agent-based agglomerative approach, and in order to decide which clusters to combine a measure of similarity between data, and sets of data has to be considered. In most of the cases, as in our case too, this is carried out by the use of an appropriate distance function and a linkage criterion. The latter one in our approach is trivial, every two observations that are closer according to the distance applied and to a similarity rate are combined. However choosing the appropriate metric is not effortless. Although the most common distance measure used is the Euclidean distance [17] , it is not the best choice for every dataset.

A. *Agent based clustering algorithm*

In this section, we provide a Pseudocode description of our agent-based clustering approach.

The application starts with initializing the system parameters — reading the dataset and normalizing it. Afterwards the agents are created and placed in their cluster, one agent per cluster.

1) Hierarchical clustering: In Algorithm 1, the main clustering procedure is described.

Algorithm 1: Agglomerative Agent Clustering

```
1: Input: $name, i_{max}, \sigma, \lambda$
2: Output: $clusters$
3: $agents = initialize\_agents(name, \sigma)$
4: for all $i \in range(0, i_{max})$ do
5:     for all $agent \in agents$ do
6:         $cluster = get\_cluster(agent)$
7:         $similar\_agent =$
8:           $search\_for\_similar(cluster, \lambda, agent.data, agent.sim\_limit)$
9:         if $similar\_agent != null$ then
10:            $new\_cluster = get\_cluster(similar\_agent)$
11:            $change\_cluster(agent, cluster)$
12:        end if
13:    end for
14: end for

end
```

The input parameters from Algorithm 1 are: $name$, i_{max}, σ, λ. The parameter $name$ denotes the name of the considered dataset (this has to be provided in a csv file in the input folder, for example "cancer.csv"). The parameter i_{max} represents the number of iterations i.e. how many times the clustering will be done by each of the agents. The similarity rate σ represents a real value (for example 0.012), that is the maximum value of the distance between data which will be allowed between two similar agents. Finally λ is the number of trials to find a similar agent, using the metric and the similarity rate provided (for example 100). The complexity of the algorithm is $\theta(i_{max} \times |agents|)$, where i_{max} denotes the number of iterations to be done, and $|agents|$ is the number of agents, i.e. the number of instances in the dataset provided. To have a good clustering the parameter i_{max} generally is larger than the cardinality of the dataset, which means that the algorithm is at least of linear complexity. In the worst case the complexity is $O(n^2)$, but by parallelizing the clustering done by the agents, it can work in linear time.

The $search\_for\_similar$ function/procedure is presented in Algorithm 2:

Algorithm 2: $search\_for\_similar$

```
1: Input: $cluster, \lambda, data, similarity\_limit$
2: Output: $similar\_agent$
3: $total\_other\_agent\_nr = get\_other\_agent\_nr()$
4: $other\_agent\_nr = random\_uniform()$
5: $other\_cluster = get\_cluster(other\_agent\_nr)$
6: if $other\_cluster != cluster$ then
7:     $other = get\_data(other\_agent\_nr)$
8:     $similarity = compute\_similarity(data, other\_data)$
9:     if $similarity <= similarity\_limit$ then
10:        $similarity\_limit = similarity$
11:        return $other\_agent\_nr$
12:    else
13:        return $search\_for\_similar(cluster, \lambda - 1, data, similarity\_limit)$
14:    end if
15: else
16:    return
17:      $search\_for\_similar(cluster, \lambda, data, similarity\_limit)$
18: end if

end
```

As we can see the other agent is searched randomly, and if it was found we compute the similarity between the current agent and the other one, and when they are similar we change the limit and return the similar agent. So, practically this algorithm searches for the most similar agent to the current agent. The similar limit initially is sigma, the parameter given by the user, and it decreases as more similar agents are found. This is necessary because we do not want an agent to find a very similar agent, and then to ”throw” it away for another one that is similar. Practically every agents searches for the most similar agent to him from the whole dataset. The $compute\_similarity$ is based on the distance applied. If lambda reaches 0 the algorithm stops and returns null.

2) Similar cluster unification: The next step is to merge the similar clusters. This is done by using representatives for each cluster. Representatives are instances with the average data from each cluster. The average is computed by the formula:

$$average = \frac{\sum\limits_{agent \in cluster} data_{agent}}{|cluster|} \tag{3}$$

This step is necessary to combine those clusters that are similar, but the data in them are not the most similar, as the idea in the previous step was.

Algorithm 3: Similar cluster unification

```
1: $representatives = create\_cluster\_representatives()$
2: for all $representative \in representatives$ do
3:     if $@\ representative\ is\ similar\ with\ cluster\ from\ clusters$ then
4:         $cluster = cluster \bigcup \{representative\}$
5:         $@\ change\ cluster\ for\ each\ data\ from\ the\ representative's\ cluster$
6:     else
7:         $clusters = clusters \bigcup new\_cluster(representative)$
8:     end if
9: end for
```

end

3) Outlier elimination: The last component is to search for outliers and to combine them with the closest cluster. Outliers are clusters that have relatively small number of instances (like 5% or less of the whole dataset).

Algorithm 4: Outlier elimination

```
1: $representatives = build\_representatives()$
2: $\{outliers, clusters\} = detect\_outliers(representatives)$
3: $new\_clusters = join\_outliers(outliers, clusters)$
```

end

The function $detect\_outliers$ separates the outliers from the real clusters (this is done relatively to a given percentage, i.e. those clusters that have smaller number of data are outliers, the other ones being the clusters we are searching for). The procedure $join\_outliers$ unifies each outlier with the most similar cluster (by the means of the metric chosen). This means that the algorithm can detect outliers, and also sorts them out.

B. Distances

In this section we present 3 metrics used in the proposed approach: Euclidean distance, Manhattan distance and Chebyshev distance [17], [18], [19]. Actually all of them are a particular case of the Minkowski distance [20]. Its formula for two vectors p,q is:

$$d(p,q) = \sqrt[p]{\sum_{i=1}^{n} |q_i - p_i|^p} \tag{4}$$

It is used typically with p being 1 or 2. The case for 1 represents the Manhattan distance and for 2 the Euclidean distance. The Chebyshev metric is the instance when p is reaching positive infinity.

1) Euclidean Distance: The Euclidean distance is the distance between two points in the Euclidean space, i.e. the length of the line segment connecting them. In Cartesian Coordinates [21] if p=(p_1, p_2, ..., p_n) and q=(q_1, q_2, .. , q_n) then the distance d is given by the Pythagorean formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{5}$$

In our algorithm we actually used the squared Euclidean distance(the square of the Euclidean distance), because we just needed to compare them and it takes less processing.

2) Manhattan Distance: The Manhattan distance or the taxicab distance [18] between two points in Cartesian Coordinate System is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. Expressed by mathematical formula:

$$d(p,q) = \sum_{i=1}^{n} |q_i - p_i| \tag{6}$$

In our design we also divided the result given by the metric with the number of the dimensions to have the final result in the interval $[0, 1]$.

3) Chebyshev Distance: The Chebyshev distance between two points is the greatest of their differences along any coordinate dimension. The formula in the standard coordinate system(Cartesian Coordinate System) is:

$$d(p,q) = \max_{1<=i<=n} |q_i - p_i| \tag{7}$$

VI. Methodology

This section presents the needed steps and infrastructure for applying our clustering approach.

A. Data sets and preprocessing

Each dataset has many instances with many attributes, so our idea was to get the datasets from csv [22] files provided by the user. Until now we can only cluster datasets with numerical data, without the class labels, and without missing data.

For preprocessing the data there are several methods that can be used [14]

- Min-max normalization: this normalization is done by using minimum and maximum values of each attribute. If min_A is the minimum and max_A is the maximum, the formula looks like:

$$value'_i = \frac{value_i - min_A}{max_A - min_A} new\_diff + new\_min_A \tag{8}$$

where $new\_diff = new\_max_A - new\_min_A$. This normalization is good to normalize all the attributes in the interval $[new\_min_A, new\_max_A]$.

- z-score normalization: based on the average mean and the standard deviation:

$$value'_i = \frac{value_i - \bar{A}}{\sigma_A} \tag{9}$$

where $\bar{A}$ is computed as:

$$\bar{A} = \frac{\sum_{i=1}^{n} value_i}{n} \tag{10}$$

and σ_A with the formula:

$$\sigma_A = \sqrt{\frac{\sum_{i=1}^{n}(value_i - \bar{A})^2}{n-1}} \tag{11}$$

This method is useful when the minimum or the maximum is unknown or in the case when some outliers are not spotted by the min-max normalization.

- normalization by decimal scaling normalizes by moving the decimal point of the value of the attribute. It is calculated as follows:

$$value'_i = \frac{value_i}{10^j} \tag{12}$$

where j is the smallest integer such that $max(|value_i|) < 1$

For our case we used the min-max normalization, with new_min_A = 0 and new_max_A = 1, because we wanted to bring all the values into the range $[0, 1]$. The formula for this particular case is:

$$value'_i = \frac{value_i - min_A}{max_A - min_A} \tag{13}$$

B. Clustering

After the whole clustering is done, we put the results in a "results" folder and identify the new result file by the exact date and time and by the dataset it was applied on. For verifying the clusters obtained, we use Elixir scripts (one for each dataset) also implemented by us in order to make the evaluation easier. The job of this scripts is to iterate through all the instances and checks if it was classified correctly or not. Also they compute statistics too, like true positives, instances in every class, accuracy, etc.

TABLE I. SEEDS DATASET RESULTS

Distance	Measure	Clusters		
Euclidean		Kama	Rosa	Canadian
	C	62	63	66
	IC	11	2	6
	Total	73	65	72
	Accuracy	88.57%	90.00%	94.28%
Manhattan		Kama	Rosa	Canadian
	C	53	60	65
	IC	15	3	14
	Total	68	63	79
	Accuracy	75.71%	85.71%	92.86%
Chebyshev		Kama	Rosa	Canadian
	C	63	65	60
	IC	15	2	5
	Total	78	67	65
	Accuracy	90.00%	92.86%	85.71%

VII. EXPERIMENTS

In Tables I,II,III we display the results obtained by applying the algorithm on three standard datasets: Seeds [11] , Cancer [9] , and Iris [4] and using the three distance functions mentioned above: Euclidean [17] , Manhattan [18] and Chebyshev [19].

For evaluating the results we provided the following measures: the number of correct instances in the cluster(C), the number of incorrect observations in the cluster(IC), the total number of data in the cluster(Total), and the precision(Prec.) calculated from the following formula:

$$accuracy = \frac{C}{nr.\ of\ instances\ in\ the\ real\ cluster} 100\ [\%] \tag{14}$$

It is important to notice that the precision is not calculated using the total number of data in the cluster, but the number of correct instances that should be in that cluster(this number is specified in the dataset description for every dataset).

A. Case study — seeds dataset

The Seeds dataset [11] contains kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian. Every wheat has 70 instances, so 210 in total. Every kernel has 7 attributes, mainly physical characteristics about them, as area, perimeter, length of kernel, etc.

Testing on seeds dataset was done with the parameters:

- Euclidean: i_{max}=200, σ=0.022, λ=200
- Manhattan: i_{max}=10, σ=0.15, λ=10
- Chebyshev: i_{max}=100, σ=0.1, λ=100

In this subsection we compare the results obtained by applying various distances on the seeds [11] dataset. These are illustrated in the Table I.

We can see that the performance of the 3 metrics is actually close, overall the Euclidean distance is the best, but the Chebyshev made the best results for two clusters. The Manhattan is a bit weaker, but it took shorter time to achieve this result.

TABLE II. CANCER DATASET RESULTS

Distance	Measure	Clusters	
Euclidean		Malignant	Benign
	C	218	450
	IC	8	23
	Total	226	473
	Accuracy	90.45%	98.25%
Manhattan		Malignant	Benign
	C	228	438
	IC	20	13
	Total	248	451
	Accuracy	94.61%	95.63%
Chebyshev		Malignant	Benign
	C	63	260
	IC	198	178
	Total	261	438
	Accuracy	26.14%	56.77%

TABLE III. IRIS DATASET RESULTS

Distance	Measure	Clusters		
Euclidean		Iris-setosa	Iris-versicolor	Iris-virginica
	C	50	49	46
	IC	0	4	1
	Total	50	53	47
	Accuracy	100%	98%	92%
Manhattan		Iris-setosa	Iris-versicolor	Iris-virginica
	C	50	50	27
	IC	0	23	0
	Total	50	73	27
	Accuracy	100%	100%	54%
Chebyshev		Iris-setosa	Iris-versicolor	Iris-virginica
	C	50	40	47
	IC	0	3	10
	Total	50	43	57
	Accuracy	100%	80%	94%

Dataset	Euclidean	Manhattan	Chebyshev	Related Work
Iris	**96.7%**	84.7%	91,3%	91.3%
Cancer	**95.6%**	95.3%	46.2%	**95.6%**
Seeds	91.0%	84.8%	89.5%	**91.9%**

TABLE IV. COMPARISON TO RELATED WORK.

B. Case study — cancer dataset

The Cancer dataset [9] consists of data, reported by a doctor from Wisconsin. It has 699 instances, every one with 9 attributes. All the attributes are between 1 and 10. The prediction has to be made upon the fact that the patient is benign or malignant. The benign class contains 458 instances and the malignant one has 241.

The parameters used were:

- Euclidean: i_{max}=100, σ=0.3, λ=100
- Manhattan: i_{max}=10, σ=0.2, λ=10
- Chebyshev: i_{max}=200, σ=0.12, λ=200

This part shows the comparison of the results obtained by applying the distances on the cancer [9] dataset. These can be seen on the table II

On this dataset also the Euclidean distance was the best, but for guessing the malignant patients the Manhattan metric is better. The Chebyshev metric did not produce anything useful.

C. Case study — iris dataset

The Iris dataset [4] dataset is one of the most popular datasets, especially in pattern recognition. Its data comes from measurements made on flowers named ”iris”. It has 150 instances, 50 for each cluster. The first class is easily separable from the other 2, however these two are harder to separate. The prediction has to be made on the class of the plant. Every instance has four attributes.

The parameters for this dataset were:

- Euclidean: i_{max}=350, σ=0.012, λ=350
- Manhattan: i_{max}=100, σ=0.045, λ=100
- Chebyshev: i_{max}=10, σ=0.2, λ=10

Here is a table with the results obtained by using the three distances mentioned on the iris [4] dataset. The Table III is used to show these.

We can see that the Euclidean distance outperforms the other too, but each cluster is done the best by a different metric. One more thing to observe is that the Chebyshev metric needed a short time to achieve this, also the accuracy for this metric is reduced.

D. Discussion

1) Analysis: As we can see the Euclidean distance works pretty well on all the datasets. However the Manhattan does not perform well on Iris and the Chebyshev does not give good results on the Cancer dataset. The euclidean distance rarely gives the best results in this benchmark for a cluster compared to the other distances, but it gives the best results on the whole datasets, that we considered. Another idea to mention is that the Chebyshev distance needs much more time and computation to converge, and also it has much lower accuracy compared to the other two metrics.

2) Comparison to related work: In Table VII-D2 we show a summary of our results in comparison with the results obtained in [10] and [12]. Columns $Euclidean$, $Manhatan$, $Chebyshev$ represent the results obtained using the metric; The last column, $Relatedwork$, represents the results obtained in [10] for datasets iris and cancer, and in [12] for seeds.

On the dataset Iris in the related paper the best result mentioned is 91.3% accuracy. In our case with metric Euclidean we have 96.7%, with the Chebyshev one 91,3%, but actually the result of applying the Manhattan metric is the only one having 100% accuracy for more than one cluster, although there is a bad accuracy for the third one. On the Cancer the Manhattan produced much better results, having an accuracy of 95.3%, and it is just a bit worse than the accuracy coming from the Euclidean distance and the accuracy of the algorithm from the approximate distance algorithm, both having 95.6%. In this case the Chebyshev metric's result is far away from what we expected, having an abysmal accuracy of 46.2%. Finally when we talk about the Seeds set, we have to mention the Gradient Algorithm, proposed by the polish guys, having an accuracy of 91.9% while our approach with Euclidean having 91.0% and with Chebyshev 89.5, so they are not that far away.

VIII. Conclusion

As we outlined in our experiments, hierarchical clustering algorithms highly depend on the metric chosen. In conclusion the Euclidean distance function is a safe choice almost for every dataset, but if we want the results to be as close to 100% as possible we should find the metric which best fits the given dataset. In the future, we consider adopting the Mahalanobis [23] distance too, which is considered to be better than the euclidean one.

References

[1] K. d. Queiroz and D. A. Good, "Phenetic clustering in biology: A critique," *The Quarterly Review of Biology*, vol. 72, no. 1, pp. pp. 3–30, 1997. [Online]. Available: http://www.jstor.org/stable/3036809

[2] "http://tinyurl.com/obj5cqd."

[3] "http://crdd.osdd.net/clusters.php."

[4] M. L. R. Iris, "http://archive.ics.uci.edu/ml/datasets/iris," 1988.

[5] Y. Shoham and K. Leyton-Brown, *Multiagent Systems*, 2009.

[6] Erlang, "http://www.erlang.org/."

[7] Elixir, "http://elixir-lang.org/."

[8] D. Thomas, *Programming Elixir*. Dallas, Texas; Raleigh, North Carolina: The Pragmatic Bookshelf, 2014.

[9] B. C. Dataset, "Machine learning repository, http://archive.ics.uci.edu/ml/ datasets/ breast+cancer+wisconsin."

[10] A. H. Cannon, L. J. Cowen, and C. E. Priebe, "Approximate distance classification," Department of Mathematical Sciences The Johns Hopkins University.

[11] M. L. R. Seeds, "https://archive.ics.uci.edu/ml/datasets/seeds," 2012.

[12] P. A. K. P. K. S. . Magorzata Charytanowicz, Jerzy Niewczas and S. Zak, "A complete gradient clustering algorithm for features analysis of x-ray images."

[13] K-means, "http://tinyurl.com/oe36koj."

[14] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[15] Hierarchical, "http://nlp.stanford.edu/ir-book/html/htmledition/hierarchical-clustering-1.html."

[16] Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, pp. 51–92, 1993.

[17] P. E. Black, "http://xlinux.nist.gov/dads//html/euclidndstnc.html," 2004.

[18] Manhattan, "http://tinyurl.com/ooczjxd."

[19] Chebyshev, "http://tinyurl.com/o2oj6le."

[20] Minkowski, "http://tinyurl.com/pmbsnyw."

[21] Cartesian, "http://tinyurl.com/cm9xc9n."

[22] CSV, "http://edoceo.com/utilitas/csv-file-format."

[23] Mahalanobis, "http://tinyurl.com/qj4shzu."