

Zdravko Markov, Ph.D.
Associate Professor of Computer Science
Central Connecticut State University

Phone: (860) 832-2711
E-mail: markovz@ccsu.edu
URL: <http://www.cs.ccsu.edu/~markov/>

Sabbatical Leave Report

Fall 2013

Contents

Sabbatical Project	2
Conclusion	5
References	5
Appendix 1.....	7
Appendix 2.....	17

Sabbatical Project

According to the sabbatical leave proposal the project included three major steps: (1) Evaluating and choosing algorithms for attribute selection and clustering, (2) Creating a software suite implementing efficiently the selected algorithms, (3) Collecting/creating data sets, exercises and laboratory projects.

During Step 1 a number of algorithms were evaluated and their performance compared to the algorithms I developed earlier. A preliminary evaluation of these algorithms was already done and the results were published in a paper I presented at the FLAIRS-26 conference [3]. During the sabbatical leave I performed a more extensive experimental evaluation using a large collection of data sets that I created for this purpose. The data sets are summarized in Table 1.

Data set	Num Attributes (Numeric)	Num Instances	Num Classes
anneal	39 (6)	898	5
arrhythmia	280 (206)	452	13
autos	26 (15)	205	6
balance-scale	5 (4)	625	3
breast-w	10 (9)	699	2
colic	23 (7)	368	2
covertypes	55	581012	7
credit-a	16 (6)	690	2
credit-g	21 (7)	1000	2
departments-small	7 (0)	20	2
departments	611 (0)	20	2
ecoli	8 (7)	336	8
glass	10 (9)	214	6
heart-statlog	14 (13)	270	2
hepatitis	20 (6)	155	2
hypothyroid	30 (7)	3772	4
ionosphere	35 (34)	351	2
iris	5 (4)	150	3
labor	17 (8)	57	2
lymph	19 (3)	148	4
reuters-2class	2887 (0)	927	2
reuters-3class	2887 (0)	1146	3
reuters	2887 (0)	1504	13
segment	20 (19)	2310	7
sick	30 (7)	3772	2
sonar	61 (60)	208	2
soybean-small	36 (0)	47	4
soybean	36 (0)	683	19
vehicle	19 (18)	846	4
vote	17 (0)	435	2
vowel	11 (10)	990	11
weather	5 (2)	14	2
webkb.binary	24080 (0)	8282	7
webkb.counts	24080 (24079)	8282	7
webkb.tfidf	24080 (24079)	8282	7

Table 1. Data sets for evaluating clustering algorithms

Most of the data sets were taken from the UCI Machine Learning repository [4] and adapted to the Weka data format. The department and department-small data sets are part of the companion website of the book [1]. I created the “webkb” files using the 4 University Data Set provided by the Web->Kb project of the CMU text learning group (<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/>). The process of creating and using these data sets for clustering is described in the lab project “Web Document Clustering”, which is also an outcome of this sabbatical project (Appendix 2).

As shown in Table 1 (Num Attributes column) many of the datasets include numeric attributes. My original idea was to convert the numeric attributes to nominal by using the standard discretization tools of Weka [2]. During the experiments however it appeared that the MDL approach may be directly used for discretization of numeric attributes. For this purpose I developed an algorithm, called MDLdiscretize. The basic idea of this algorithm is the following. We order the set of values of the numeric attribute (excluding repetitions) and look for a break point, which splits them into two intervals. We consider each value as a possible break point and compute the MDL for the split of data based on this value, and then choose the one that minimizes MDL. I implemented the algorithm in Java and performed experiments with the data sets which include numeric attributes. I also incorporated the MDLdiscretize algorithm in the attribute selection and clustering algorithms (MDLraker and MDLcluster), which were originally designed for nominal attributes only. In this way these algorithms can now be used with nominal, numeric or mixed data. The MDLdiscretize algorithm is briefly described in the manual for the MDL clustering suite (<http://www.cs.ccsu.edu/~markov/MDLclustering/MDLmanual.pdf>), which is also included in Appendix 1.

To evaluate the performance of the newly developed MDLdiscretize algorithm and the extended MDLcluster and MDLranker algorithms I run experiments with the collection of datasets I have created. I used the following experimental setting. On each data set I ran the k-Means and EM algorithms using the Weka system [2], and the extended MDLcluster algorithm. I didn't use the MDLranker algorithm separately, because it is incorporated in MDLcluster (at each level of the clustering tree it splits the data into clusters by using the attribute that minimizes MDL, the same approach used for attribute ranking). As K-Means and EM use random choice for the initial clustering I ran them several times with different seeds for the random number generator and reported the run that maximized their performance. The number of clusters is a parameter for k-Means and EM and was set to the known number of classes (shown in column 3 of Table 1). MDLcluster produces a clustering tree with number of leaves depending on a cutoff parameter setting a threshold for the information compression in a cluster. This parameter was experimentally set so that the number of clusters (leaves) was as close as possible to the known number of classes.

To compare the MDL discretizaion independently from clustering I applied MDLdiscretize and then ran k-Means and EM on the discretized data. As all datasets in the collection are labeled I used the Weka's classes-to-clusters accuracy as a performance measure. It is a comparison to the “true” cluster membership specified by the class

attribute and is computed as the total number of majority labels in the leaves of the clustering tree divided by the total number of instances in data. To investigate the effect of discretization on the mapping between attribute space and class labels I also run a decision tree classifier (Weka's J48 algorithm) on the original data and on the MDL discretized data. The results of these experiments are summarized in Table 2.

Data set	k-Means	EM	MDLcluster (# of cls)	k-Means (MDL)	EM (MDL)	J48	J48 (MDL)
anneal	64	64	76 (9)	60	48	98	96
arrhythmia	27	40	55 (13)	23	26	64	56
autos	43	46	56 (22)	43	49	82	80
balance-scale	55	53	68 (3)	57	64	77	70
breast-w	96	94	84 (2)	94	94	95	95
colic	76	65	71 (5)	73	65	85	85
covertype	33	38	48 (8)	32	33	94	74
credit-a	79	74	66 (14)	78	73	86	85
credit-g	63	64	70 (11)	63	63	71	71
departments-small	90	85	80 (2)	-	-	-	-
departments	65	60	70 (2)	-	-	-	-
ecoli	63	78	71 (8)	56	70	84	72
glass	44	45	56 (6)	50	47	67	69
heart-statlog	80	81	62 (2)	80	80	77	83
hepatitis	74	74	79 (2)	79	76	84	81
hypothyroid	50	65	92 (4)	61	48	99	96
ionosphere	71	75	74 (2)	62	64	91	88
iris	89	91	78 (3)	60	65	96	88
labor	89	93	80 (2)	84	95	74	86
lymph	53	76	62 (8)	56	66	77	78
reuters-2class	80	72	87 (2)	-	-	-	-
reuters-3class	50	60	71 (3)	-	-	-	-
reuters	39	43	57 (13)	-	-	-	-
segment	67	61	67 (8)	60	61	97	83
sick	88	82	93 (2)	88	84	99	94
sonar	56	54	53 (2)	61	58	71	76
soybean-small	100	100	100 (7)	-	-	-	-
soybean	51	67	54 (20)	-	-	-	-
vehicle	39	37	50 (4)	42	41	72	61
vote	88	88	67 (2)	-	-	-	-
vowel	35	39	32 (11)	30	33	81	62
weather	71	67	64 (2)	71	65	64	57
webkb.binary	41		46 (8)	-	-	-	-
webkb.counts			48 (5)				
webkb.tfidf			46 (8)				

Table 2. Classes-to-cluster accuracies (%) for k-Means, EM, and MDLcluster, and 10-fold cross validation accuracies for J48.

With most of the data sets MDLcluster achieved better accuracy than k-Means and EM. A very important advantage of MDLcluster over k-Means and EM was its substantially lower run time and memory usage. In average the MDLcluster run time was 10 times smaller than k-Means and EM. The webkb datasets, which are really large, took about 1.5 hours each for MDLcluster, while k-Means and EM could not complete because of

insufficient memory or more than 24 hours runtime. The results reported in the table are obtained with a 10% random sample of the original data. The experiments were performed on a 3GHz Intel-based PC with 4GB memory.

The accuracies produced by k-Means and EM on the MDL discretized data were similar to those on the original data. This shows that the 2-bin MDL discretization preserves the structure of the attribute space by providing good density estimation of the numeric attributes and at the same time considerably simplifying the data and the clustering models. Similar conclusion may be drawn analyzing the results obtained from the J48 classifier experiments. As expected, the classification accuracy on the discretized data was a little lower (because of the loss of information in the two-bin density estimation), but the classification models were substantially smaller and more comprehensible. For example, the J48 model for the “covertime” data has 19229 nodes, while the tree obtained from the MDL discretized data has 2343 nodes.

Conclusion

The results of the sabbatical project can be summarized as follows:

1. A new Machine Learning algorithm for attribute discretization (MDLdiscretize) was developed and two previous algorithms (MDLcluster and MDLranker) were extended to work with numeric data. The algorithms were experimentally evaluated and compared with state-of-the-art algorithms in the field. A brief description of the MDLdiscretize algorithm is included in the MDL clustering manual (Appendix 1).
2. All three algorithms were efficiently implemented in Java and made publicly available on the web as a software suite called MDL Clustering (<http://www.cs.ccsu.edu/~markov/MDLclustering/>). The manual for using this software is included in Appendix 1.
3. A large collection of datasets suitable for clustering and classification experiments was created. The datasets are included in the MDL Clustering suite.
4. A lab project for Web document clustering was developed. It is included in the MDL Clustering suite and also attached in Appendix 2.
5. The developed algorithms and the obtained experimental results provide good material for a scientific publication. I’m currently working on a paper that will be submitted to a major Machine Learning or Data Mining conference.

I find the time spent on my sabbatical project enriching and productive. The results I obtained largely contribute to my personal development as a teacher and scholar. The software I developed benefits students and faculty at CCSU, especially those from the two graduate programs – Computer Information Technology and Data Mining. It is also a general contribution in the field of Machine Learning and Data Mining.

References

1. Zdravko Markov and Daniel T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Wiley, April 2007.

2. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten. The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1, 2009. (<http://www.cs.waikato.ac.nz/ml/weka/>).
3. Zdravko Markov. MDL-based Unsupervised Attribute Ranking, Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS-26), St. Pete Beach, Florida, USA, May 22-24, 2013, AAAI Press 2013, pp. 444-449. Full text in PDF available from AAAI at <http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5845/6115>.
4. Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science.

Appendix 1

Java Classes for MDL-Based Attribute Ranking and Clustering

Zdravko Markov
Computer Science Department
Central Connecticut State University
New Britain, CT 06050, USA
<http://www.cs.ccsu.edu/~markov/>

1. Introduction	8
2. Attribute Ranking	8
3. Attribute Discretization	10
4. Clustering	11
5. Utilities	14
5.1. ARFFstring class	14
5.2. Increasing the heap size for the Java virtual machine	15
6. References	16

1. Introduction

This document describes algorithms for attribute ranking and clustering originally described in [1, 2], an algorithm for attribute discretization briefly outlined hereafter, and a utility for creating string ARFF files. The algorithms are available as Java classes from a JAR file, which can be downloaded at <http://www.cs.ccsu.edu/~markov/MDLclustering/MDL.jar>. The archive also includes all Weka classes [3]. The archive is executable and starts the Weka command-line interface (Simple CLI), which provides access to these algorithms and to the full functionality of the Weka data mining software as well (see the Weka manual at <http://www.cs.waikato.ac.nz/ml/weka/documentation.html>). The data files mentioned in this document are available from a zip archive at <http://www.cs.ccsu.edu/~markov/MDLclustering/data.zip>.

2. Attribute Ranking

The original idea of the attribute ranking proposed in [1, 2] is the following. First we split the data using the values of an attribute, i.e. create a clustering such that each cluster contains the instances that have the same value for that attribute. Then we compute the MDL of this clustering and use it as a measure to evaluate the attribute assuming that the best attribute minimizes MDL. This approach works for nominal attributes. To apply it to numeric attributes we first split the range of the attribute values into two intervals, which then can be used as nominal values for ranking. The idea is to find a breakpoint that minimizes the MDL of the resulting split of data. Hereafter we illustrate this process with the weather data example (data file weather.arff).

outlook	temperature	humidity	windy	play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rainy	71	91	TRUE	no

Two of the attributes in this dataset are numeric – temperature and humidity. Assume we want to evaluate humidity. First we order the values of this attribute (excluding repetitions) and start looking for a breakpoint, which will split them in two subsets. We

consider each value as a possible breakpoint and compute the MDL for the split of data based on this value. When computing MDL we exclude the class attribute (play). The result of this process is shown in the table below, where the first row shows the ordered values (x), the second – the distribution of instances based on the split of data using that value (the number of instances with humidity less than or equal to x / the number of instances with humidity greater than x), and the third – the MDL for the split.

x	65	70	75	80	85	86	90	91	95	96
$\#[65, x] / \#(x, 96]$	1 / 13	4 / 10	5 / 9	7 / 7	8 / 6	9 / 5	11 / 3	12 / 2	13 / 1	–
MDL($[65, x], (x, 96]$)	201.70	203.77	203.61	200.48	201.28	202.03	205.65	204.04	201.70	–

Choosing a value of x as a breakpoint defines a possible discretization of the attribute in two intervals, $[65, x]$ and $(x, 96]$. The best split is the one that minimizes MDL, which is obtained with breakpoint 80, or the intervals $[65, 80]$ and $(80, 96]$. Thus we compute the MDL of humidity as 200.48 and then rank it along with the rest of the attributes.

The general format for running this algorithm in CLI is:

```
java MDLranker <input file>.arff [# of attributes] [<output
file>.{arff|csv}] [de]
```

The input file must be in ARFF format. The second argument specifies the number of attributes from the top of the ranked list to be displayed or written to the output file. Note that the last attribute in the data set (the default class attribute, “play” in the weather data) is not used for ranking and is excluded from this list (but added in the output file). The output file format may be either ARFF or CSV and is determined by the file name extension. When the fourth argument (de) is specified the numeric attributes are first transformed by applying nonparametric density estimation. For this purpose we use equal-interval binning and calculate the width of the interval h using the Scott’s method as $h = \frac{3.5\sigma}{n^{1/3}}$, where n is the number of values of the attribute and σ – its standard deviation. Below is the output produced by the MDLranker algorithm applied to the weather data. The data set is written on the output file weather.ranked.csv, where the attributes humidity and temperature are switched.

```
> java MDLranker data/weather.arff 4 data/weather.ranked.csv
```

```
Attributes: 5
Ignored attribute: play
Instances: 14 (non-sparse)
Attribute-values in original data: 27
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 197.39
```

```
-----
Top 4 attributes ranked by MDL:
199.47 @attribute outlook {sunny,overcast,rainy}
200.48 @attribute humidity numeric
201.29 @attribute temperature numeric
202.51 @attribute windy {TRUE,FALSE}
-----
```

Note that the minimum encoding length of data is less than the MDL of the best attribute. This usually happens when there are too many attribute values and indicates that the MDL encoding does not help to reduce the code length of data. Applying density estimation reduces the number of attribute values and lowers the MDL encoding length as shown below.

```
> java MDLranker data/weather.arff 4 de
```

```
Attributes: 5
Ignored attribute: play
Instances: 14 (non-sparse)
Attribute-values in original data: 27
Numeric attributes with missing values (replaced with mean): 0
Attribute-values after density estimation: 9
Minimum encoding length of data: 97.68
```

```
-----
Top 4 attributes ranked by MDL:
92.07 @attribute outlook {sunny,overcast,rainy}
94.15 @attribute temperature numeric
94.15 @attribute humidity numeric
94.15 @attribute windy {TRUE,FALSE}
```

3. Attribute Discretization

The attribute discretization algorithm processes the numeric attributes in the same way as the MDLranker (described above). The general format for running this algorithm in CLI is:

```
java MDLdiscretize <input file>.arff <output file>.{arff|csv} [de]
```

The input file must be in ARFF format, while the output file format is specified by the file extension. When the third argument (de) is specified the numeric attributes are first transformed by density estimation (as in the MDLranker algorithm). Below is the output produced by the MDLdiscretize algorithm applied to the weather data:

```
> java MDLdiscretize data/weather.arff weather.discretized.arff
```

```
Attributes: 5
Ignored attribute: play
Instances: 14 (non-sparse)
Attribute-values in original data: 27
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 197.39
-----
@attribute temperature {[64-80],(80-85]}
@attribute humidity {[65-80],(80-96]}
2 numeric attributes discretized
Time(ms): 31
```

The contents of the output file is the following:

```

@relation weather.MDLdiscretize

@attribute outlook {sunny,overcast,rainy}
@attribute temperature {[64-80],(80-85]}
@attribute humidity {[65-80],(80-96]}
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}

@data
sunny, (80-85], (80-96], FALSE, no
sunny, [64-80], (80-96], TRUE, no
overcast, (80-85], (80-96], FALSE, yes
rainy, [64-80], (80-96], FALSE, yes
rainy, [64-80], [65-80], FALSE, yes
rainy, [64-80], [65-80], TRUE, no
overcast, [64-80], [65-80], TRUE, yes
sunny, [64-80], (80-96], FALSE, no
sunny, [64-80], [65-80], FALSE, yes
rainy, [64-80], [65-80], FALSE, yes
sunny, [64-80], [65-80], TRUE, yes
overcast, [64-80], (80-96], TRUE, yes
overcast, (80-85], [65-80], FALSE, yes
rainy, [64-80], (80-96], TRUE, no

```

Using the density estimation option changes the intervals for the values of temperature and humidity:

```
> java MDLdiscretize data/weather.arff weather.discretized.arff de
```

```

Attributes: 5
Ignored attribute: play
Instances: 14 (non-sparse)
Attribute-values in original data: 27
Numeric attributes with missing values (replaced with mean): 0
Attribute-values after density estimation: 9
Minimum encoding length of data: 97.68
-----
@attribute temperature {[64.0-74.5],(74.5-85.0]}
@attribute humidity {[65.0-80.5],(80.5-96.0]}
2 numeric attributes discretized
Time(ms): 125

```

4. Clustering

The MDL clustering algorithm is described in [1]. It starts with the data split produced by the attribute that minimizes MDL and then recursively applies the same procedure to the resulting splits, thus generating a hierarchical clustering. For nominal attributes the number of splits is equal to the number of attribute values. Numeric attributes are treated in the same way as in the previous algorithms, and then the breakpoint that minimizes MDL is used to split the data in two subsets. The process of growing the clustering tree is controlled by a parameter evaluating the information compression at each node. The information compression is computed as the difference between the code length of the

data at the current node of the tree and the MDL of the attribute that produces the data split. If the compression becomes lower than a specified cutoff value the process of growing the tree stops and a leaf node is created. An experimentally determined value of 20% of the information compression at the root of the tree is used as a default cutoff.

The general format for running this algorithm in CLI is the following:

```
java MDLcluster <input file>.arff [compr. cutoff] [de] [<output
file>.{arff|csv}]
```

The input file must be in ARFF format. When specified the output file (in ARFF or CSV format) contains the cluster assignments as an additional attribute added as a last attribute (after the class attribute). If the compression cutoff is omitted the default value is used (20% of the initial compression). The meaning of the **de** argument is the same as in the previous algorithms.

An example of running MDLcluster with the iris data set is shown below:

```
> java MDLcluster data/iris.arff

Attributes: 5
Ignored attribute: class
Instances: 150 (non-sparse)
Attribute-values in original data: 123
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 3467.11
-----
(238.38) (47.68)
#petallength<=1.9 (49.44)
  #sepalwidth<=3.2 (10.37) [18,0,0] Iris-setosa
  #sepalwidth>3.2 (22.63) [32,0,0] Iris-setosa
#petallength>1.9 (78.29)
  #sepalwidth<=6.2 (28.20) [0,36,13] Iris-versicolor
  #sepalwidth>6.2 (30.90) [0,14,37] Iris-virginica
-----
Number of clusters (leaves): 4
Correctly classified instances: 123 (82%)
Time(ms): 16
```

The number in the parentheses represents the information compression at the corresponding node of the clustering tree (the second number at the root is the default cutoff), and the numbers in square brackets – the distribution of the class labels at the tree leaves. For each leaf the majority class label is also shown. The class distribution in the leaves provides information for evaluating the clustering quality when class labels are known (but ignored for the purposes of clustering) by using the classes-to-clusters evaluation measure (also used in Weka). It is a comparison to the “true” cluster membership specified by the class attribute and is computed as the total number of majority labels in the leaves divided by the total number of instances. This measure is reported as “Correctly classified instances” (123 out of 150, or 82%).

Below is another example that illustrates the use of the compression cutoff.

```
> java MDLcluster data/soybean-small.arff
```

```
Attributes: 36
Ignored attribute: class
Instances: 47 (non-sparse)
Attribute-values in original data: 72
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 3221.60
-----
```

```
(951.79) (190.36)
stem-cankers=0 (51.15) [0,10,0,0] D2
stem-cankers=1 (153.77) [0,0,10,8] D3
stem-cankers=2 (53.56) [0,0,0,9] D4
stem-cankers=3 (46.84) [10,0,0,0] D1
-----
```

```
Number of clusters (leaves): 4
Correctly classified instances: 39 (82%)
```

This clustering is produced with the default cutoff of 190.36, so all leaves of the tree have a smaller compression. The class distribution information suggests that we may get a better clustering by splitting cluster “stem-cankers=1”, because it is not “pure” (as the other three). If we don’t know the class labels (in this case we have to use a dummy class attribute), we still may be able to arrive at the same conclusion just by comparing the compression values of different clusters – the compression of “stem-cankers=1” is substantially larger than the compression at the other clusters, so it may be possible to expand the tree at this node and still have a good compression. To do this we have to use a cutoff value less than 153.77 and greater than 53.56 (not to split any of the other clusters). The result of this run is shown below.

```
> java MDLcluster data/soybean-small.arff 100
```

```
Attributes: 36
Ignored attribute: class
Instances: 47 (non-sparse)
Attribute-values in original data: 72
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 3221.60
-----
```

```
(951.79)
stem-cankers=0 (51.15) [0,10,0,0] D2
stem-cankers=1 (153.77)
  canker-lesion=1 (78.09) [0,0,10,0] D3
  canker-lesion=2 (42.95) [0,0,0,8] D4
stem-cankers=2 (53.56) [0,0,0,9] D4
stem-cankers=3 (46.84) [10,0,0,0] D1
-----
```

```
Number of clusters (leaves): 5
Correctly classified instances: 47 (100%)
```

5. Utilities

5.1. ARFFstring class

By using the ARFFstring class large collections of text/HTML files may be transformed into ARFF files with string attributes and then converted to TF/TFIDF type by applying the StringToWordVector filter. The general format of using this class is the following:

```
java ARFFstring <input directory> <class label> <output file>
```

This command creates a text file with as many lines as files in the input directory, where each line contains the following:

```
"file name", "file content", "class label"
```

The files in the input directory must contain plain text or HTML, where all HTML tags are removed.

Below we describe the steps for creating an ARFF file from the departments document collection available from <http://www.cs.ccsu.edu/~markov/MDLclustering/data.zip>:

1. Create file deptA with the files in folder data/departments/A with class label A:

```
java ARFFstring data/departments/A A deptA
```

2. Create file deptB with the files in folder data/departments/B with class label B:

```
java ARFFstring data/departments/B B deptB
```

3. Merge deptA and deptB. This can be done in a command prompt window with the copy command:

```
copy deptA + deptB departments-string.arff
```

4. Add the following ARFF file header in the beginning of departments-string.arff:

```
@relation departments_string
@attribute document_name string
@attribute document_content string
@attribute document_class string
```

```
@data
```

5. Convert the first and the third attribute into nominal and write the output on temp1.arff:

```
java weka.filters.unsupervised.attribute.StringToNominal
-i departments-string.arff
-o temp1.arff -R 1,3
```

6. Transform the second attribute (`document_content`) into a set of numeric attributes representing the word presence (0/1) in the documents:

```
java weka.filters.unsupervised.attribute.StringToWordVector
-i temp1.arff
-o temp2.arff
-S
-tokenizer weka.core.tokenizers.AlphabeticTokenizer
```

7. Move the `document_class` attribute to the end and write the output to `departments.arff`:

```
java weka.filters.unsupervised.attribute.Reorder
-i temp2.arff
-o departments.arff
-R 1,3-last,2
```

8. The data set `departments.arff` can now be used for clustering, classification and other experiments. For example:

```
> java MDLcluster departments.arff
```

```
Attributes: 612
Ignored attribute: document_class
Instances: 20 (sparse)
Attribute-values in original data: 1234
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 24569.11
-----
(4305.58) (861.12)
#major<=0 (1498.23)
  #computer<=0 (247.68) [2,3] B
  #computer>0 (255.52) [4,1] A
#major>0 (1452.01)
  #offers<=0 (265.72) [3,2] A
  #offers>0 (264.13) [2,3] B
-----
Number of clusters (leaves): 4
Correctly classified instances: 13 (65%)
Time(ms): 375
```

5.2. Increasing the heap size for the Java virtual machine

The parameter `-Xmx<heap size>` may be used in the command line to set the maximum heap size used by the Java virtual machine. This can be done when starting the `MDL.jar` in the Windows command prompt. For example, the following command will initialize the Java virtual machine with 1600 Mb maximum heap size and start the Weka CLI.

```
java -Xmx1000m -jar MDL.jar
```

Similarly, any class from MDL.jar may be started in the command prompt window after extracting the archive. For example, the following command will initialize the Java virtual machine with 1000 Mb maximum heap size and start the MDLcluster class.

```
java -Xmx1000m MDLcluster data/weather.arff
```

6. References

1. Zdravko Markov. MDL-based Unsupervised Attribute Ranking, Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS-26), St. Pete Beach, Florida, USA, May 22-24, 2013, AAAI Press 2013, pp. 444-449.
<http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5845/6115>.
2. Zdravko Markov and Daniel T. Larose. MDL-Based Model and Feature Evaluation, in Chapter 4 of *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Wiley, April 2007, ISBN: 978-0-471-66655-4.
3. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009). The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

Appendix 2

Web Document Clustering

Lab Project based on the MDL clustering suite
<http://www.cs.ccsu.edu/~markov/MDLclustering/>

Zdravko Markov
Computer Science Department
Central Connecticut State University
New Britain, CT 06050, USA
<http://www.cs.ccsu.edu/~markov/>

1. Introduction	18
2. Data Collection	18
3. Data Preprocessing	19
3.1. Creating a string ARFF file	20
3.2. Creating the vector space model	20
4. Clustering	21
5. Attribute Ranking	23
6. Further experiments	25
7. References	26

1. Introduction

Web document clustering is an important application of Machine Learning for the Web. A clustering system can be useful in web search for grouping search results into closely related sets of documents. Clustering can improve similarity search by focusing on sets of relevant documents and hierarchical clustering methods can be used to automatically create topic directories, or organize large collections of web documents for efficient retrieval. In this lab project we illustrate the basic steps of web document clustering by using web pages collected from computer science departments of various universities by the CMU World Wide Knowledge Base (Web->Kb) project [1]. We first describe the data preprocessing steps, which use basic techniques from information retrieval. Then we apply a clustering algorithm to create hierarchies of web pages and analyze the results. A recommended reading for this project is the book “Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage” [4] – Chapters 1, 3, and 4. Chapter 1 discusses the techniques used for preprocessing of web pages. Chapter 3 describes the basic algorithms used for web document clustering, and Chapter 4 – the approaches to evaluating the clustering results. The tools we use for this project are Java implementations of data preprocessing and machine learning algorithms available from the Weka data mining system [5] extended with MDL-based algorithms for attribute ranking and clustering described in [2, 3]. The algorithms are available as Java classes from an executable JAR file, which can be downloaded at <http://www.cs.csu.edu/~markov/MDLclustering/>. This site also provides a manual describing their functionality and usage [2].

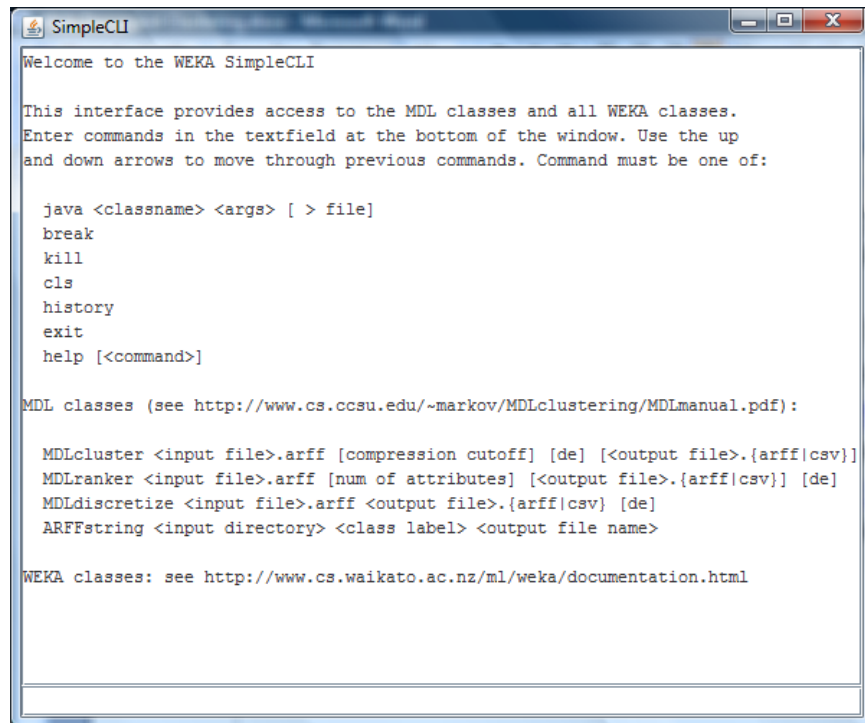
2. Data Collection

The data set we are using is described at <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/>. It contains 8,282 web pages collected from the web sites of four universities: Cornell (867), Texas (827), Washington (1205), Wisconsin (1263), and 4,120 miscellaneous pages collected from other universities. All pages are manually grouped into 7 categories: student (1641), faculty (1124), staff (137), department (182), course (930), project (504), and other (3764). The task at this step is to download the data set from <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/webkb-data.gtar.gz>, unzip it, and organize the files in folders. The latter can be done in two ways: by topic or by university. Also, we may create one data set with all data, create individual data sets for each university organized by topic, or for each topic organized by university. How we do this depends on the machine learning problem we want to solve. In this project we investigate the natural groupings of web pages based on their topic content. To minimize the processing time and balance the data we create a small subset with web pages from one university only, for example Cornell, and also exclude the department category, because it contains only one page, and the other and student categories because they have too many pages. Thus we end up with a balanced set of 4 folders containing 119 web pages: course (44), faculty (34), project (20), and staff (21).

3. Data Preprocessing

The task now is to create a vector space model of our dataset (see [4], Chapter 1). First we create a text corpus by putting all documents together and then remove punctuation, HTML tags, and short words (the so-called stop words). Then the remaining words (called terms) are used as attributes to represent the web pages. There are three possible representations: boolean, where the attribute value are 1/0 only, indicating word presence/absence; term frequency (TF), where the value of the attribute is the number of occurrences of the word in the document; and term frequency - inverse document frequency (TFIDF), where the word frequency is weighted with the frequency of the word occurrence across different documents. All three representations may be used for clustering.

The processing steps described above can be performed by using the MDL clustering suite. It is an executable JAR file, which can be executed by opening the URL <http://www.cs.ccsu.edu/~markov/MDLclustering/MDL.jar>, or by downloading it first and starting it in a command prompt. The second option allows adjustment of the heap size of the Java virtual machine (see <http://www.cs.ccsu.edu/~markov/MDLclustering/MDLmanual.pdf>). The main class in this JAR file opens a window that provides a command-line interface (CLI) to all Java classes.



```
SimpleCLI
Welcome to the WEKA SimpleCLI

This interface provides access to the MDL classes and all WEKA classes.
Enter commands in the textfield at the bottom of the window. Use the up
and down arrows to move through previous commands. Command must be one of:

  java <classname> <args> [ > file]
  break
  kill
  cls
  history
  exit
  help [<command>]

MDL classes (see http://www.cs.ccsu.edu/~markov/MDLclustering/MDLmanual.pdf):

  MDLcluster <input file>.arff [compression cutoff] [de] [<output file>.{arff|csv}]
  MDLranker <input file>.arff [num of attributes] [<output file>.{arff|csv}] [de]
  MDLdiscretize <input file>.arff <output file>.{arff|csv} [de]
  ARFFstring <input directory> <class label> <output file name>

WEKA classes: see http://www.cs.waikato.ac.nz/ml/weka/documentation.html
```

3.1. Creating a string ARFF file

The Attribute-Relation File Format (see <http://weka.wikispaces.com/ARFF>) is an input file format for the Weka data mining system and is used by all classes from the MDL clustering suite. The string ARFF file is the input to the Weka StringToWordVector filter that creates the vector space model of text documents. So, our first step is to transform the collection of 119 web pages into a string ARFF file. For this purpose we use a utility class called ARFFstring. We apply it to each folder by passing the folder name (1st argument), the class label (2nd argument), and the output file (3rd argument).

```
java ARFFstring course course temp1
java ARFFstring faculty faculty temp2
java ARFFstring project project temp3
java ARFFstring staff staff temp4
```

This sequence of commands creates four text files – temp1 through temp4. Each one has as many lines as files in the input folder, where each line contains the following:

```
"file name", "file content", "label"
```

These are values of the attributes, describing each data instance (web page). The ARFF file should start with a header, which names the relation (data set) and defines the types of these attributes (string):

```
@relation webkb_string

@attribute document_name string
@attribute document_content string
@attribute document_class string

@data
```

The header must be followed by the data instances. So, we need to merge the header and the four text files in that order. We can do this with the copy command in a Windows Command Prompt (header is a file containing the header).

```
copy header+temp1+temp2+temp3+temp4 cornell-string.arff
```

3.2. Creating the vector space model

The next steps use Weka classes, which are described in the Command-line section of the Weka manual available at <http://www.cs.waikato.ac.nz/ml/weka/documentation.html>.

Before applying the StringToWordVector class we need to perform two additional transformations. First, because the document_name attribute is not needed for the purposes of clustering or classification, we remove it by using the Remove filter and save the data in temp1.arff (the -R option specifies the attribute index).

```
java weka.filters.unsupervised.attribute.Remove
-i cornell-string.arff
-o temp1.arff
-R 1
```

Then we convert the second attribute (`document_class`) into nominal type because it will be needed for analyzing the clustering results. To do this we use the `StringToNominal` filter.

```
java weka.filters.unsupervised.attribute.StringToNominal
-i temp1.arff
-o temp2.arff
-R 2
```

Finally we apply the `StringToWordVector` filter, where we specify only three arguments, which affect the way words are converted to terms. The `-S` argument specifies that the words that occur in the stop list will be ignored, `-L` causes all words to be converted to lower case, and the tokenizer parameter specifies that only alphabetical words will be used as terms. Three other arguments, `-C`, `-T`, and `-I`, may be used to determine the type of the representation (boolean, TF, or TFIDF). If omitted, the filter uses the boolean word presence representation (0/1). A complete list of arguments can be obtained if the class is run without any arguments.

```
java weka.filters.unsupervised.attribute.StringToWordVector
-i temp2.arff
-o temp3.arff
-S
-L
-tokenizer weka.core.tokenizers.AlphabeticTokenizer
```

The output file `temp3.arff` contains the `document_class` attribute followed by 3452 attributes representing the terms. We still need one more transformation that will move the `document_class` at the end of the list of attributes, because this is the default position of the class attributes used by all clustering and classification algorithms. So, we apply the `Reorder` filter to achieve this and save the output to the file `cornell-binary.arff`, which can be used for clustering experiments.

```
java weka.filters.unsupervised.attribute.Reorder
-i temp3.arff
-o cornell-binary.arff
-R 2-last,1
```

4. Clustering

The MDL clustering algorithm is originally described in [3] and its use in the MDL clustering suite is describe in the manual [2]. The algorithm starts with the data split

produced by the attribute that minimizes MDL and then recursively applies the same procedure to the resulting splits, thus generating a hierarchical clustering. The process of growing the clustering tree is controlled by a parameter evaluating the information compression at each node. If the compression becomes lower than a specified cutoff value the process of growing the tree stops and a leaf node is created. An experimentally determined value of 20% of the information compression at the root of the tree is used as a default cutoff.

We now apply the MDL clustering algorithm to our dataset by the following command in CLI.

```
java MDLcluster cornell-binary.arff
```

The output produced by the algorithm is the following:

```
Attributes: 3453
Ignored attribute: document_class
Instances: 119 (sparse)
Attribute-values in original data: 6897
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 819944.89
-----
(136031.67) (27206.33)
#research<=0 (65596.67)
  #programming<=0 (33239.11)
    #science<=0 (13108.54) [13,1,4,2] course
    #science>0 (11940.50) [8,4,2,2] course
  #programming>0 (23159.31) [18,2,3,0] course
#research>0 (68599.79)
  #acm<=0 (37113.65)
    #system<=0 (19322.73) [1,9,5,8] faculty
    #system>0 (8759.16) [4,3,2,3] course
  #acm>0 (24892.25) [0,15,4,6] faculty
-----
Number of clusters (leaves): 6
Correctly classified instances: 67 (56%)
```

The numbers in parentheses represent the information compression at the corresponding node of the clustering tree (the second number at the root is the default cutoff), and the numbers in square brackets – the distribution of the class labels at the tree leaves. For each leaf the majority class label is also shown. The class distribution in the leaves provides information for evaluating the clustering quality when class labels are known (but ignored for the purposes of clustering) by using the classes-to-clusters evaluation measure, which is reported as “Correctly classified instances”.

Although the overall accuracy of the above clustering tree (67 out of 119, or 56%) is relatively low there are some good clusters in it. For example, the last leaf identifies faculty web pages by the presence of the terms “research” and “acm”, which makes a very good sense. The top level split also makes sense and provides almost the same classes-to-clusters accuracy. With 55% confidence we may conclude that if the page

includes the term “research” it belongs to a faculty. We can force the algorithm to stop at the first level of the tree by specifying a cutoff greater than 68599.79.

```
> java MDLcluster cornell-binary.arff 70000
...
(136031.67)
#research<=0 (65596.67) [39,7,9,4] course
#research>0 (68599.79) [5,27,11,17] faculty
-----
Number of clusters (leaves): 2
Correctly classified instances: 66 (55%)
```

5. Attribute Ranking

The data set we created is very unbalanced in terms of attributes/instances ratio (3453/119). This normally happens with text data because the number of words in text documents is usually very large compared to the number of documents in the sample. In this case we may want to reduce the number of attributes by selecting a subset that can still represent the data well. When the documents have class labels (supervised attribute selection) we may look for attributes that best preserve the class distribution. Without class labels (unsupervised attribute selection) we may select attributes that best preserve the natural grouping of data in clusters. This is the approach that we take in this study because we use the class labels only for evaluation purposes. The MDL clustering suite offers an algorithm for attribute selection based on attribute ranking. It orders attributes by their relevance to the clustering task and then by specifying a parameter we may select a number of attributes from the top of the ordered list. For example, the following command selects the best 200 attributes of our data set cornell-binary.arff:

```
java MDLranker cornell-binary.arff 200 cornell-binary200.arff
```

The command prints the specified number of attributes from the top of the ranked list and also creates a data file with these attributes (cornell-binary200.arff).

```
Top 200 attributes ranked by MDL:
683913.22 @attribute research numeric
688637.65 @attribute publications numeric
690134.95 @attribute system numeric
695502.55 @attribute time numeric
701401.95 @attribute information numeric
701445.51 @attribute programming numeric
701673.78 @attribute systems numeric
702005.01 @attribute science numeric
704228.14 @attribute university numeric
704779.31 @attribute acm numeric
705077.50 @attribute work numeric
705306.74 @attribute computing numeric
707767.91 @attribute department numeric
707782.86 @attribute software numeric
708184.77 @attribute page numeric
...
```

As expected the attribute “research” is on top of the list (it is also on the top of the clustering tree). We can now apply MDL clustering to the newly created dataset.

```
> java MDLcluster cornell-binary200.arff

Attributes: 201
Ignored attribute: document_class
Instances: 119 (sparse)
Attribute-values in original data: 400
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 47046.82
-----
(776.71) (155.34)
#materials<=0 (622.36)
  #handouts<=0 (663.86)
    #assignments<=0 (465.53)
      #cs<=0 (608.13)
        #system<=0 (350.88)
          #information<=0 (75.47) [0,0,2,2] project
          #information>0 (11.31) [0,0,3,0] project
          #system>0 (1.48) [0,1,3,0] project
        #cs>0 (493.51)
          #research<=0 (818.56)
            #computer<=0 (173.37)
              #sunday<=0 (54.51) [1,1,2,0] project
              #sunday>0 (9.75) [3,0,0,0] course
            #computer>0 (389.95)
              #ithaca<=0 (169.51)
                #department<=0 (-2.00) [1,1,1,0] course
                #department>0 (40.67) [0,3,0,1] faculty
              #ithaca>0 (41.41) [0,2,1,1] faculty
            #research>0 (330.11)
              #information<=0 (391.29)
                #software<=0 (419.46)
                  #problems<=0 (137.91) [1,4,1,1] faculty
                  #problems>0 (68.40) [0,5,0,1] faculty
                #software>0 (216.09)
                  #time<=0 (99.92) [0,3,1,4] staff
                  #time>0 (16.30) [0,2,1,2] faculty
              #information>0 (245.15)
                #email<=0 (176.46)
                  #activities<=0 (181.44)
                    #acm<=0 (76.48) [1,0,1,4] staff
                    #acm>0 (82.12) [0,6,3,1] faculty
                  #activities>0 (1.02) [0,5,0,0] faculty
                #email>0 (128.66) [0,1,1,4] staff
            #assignments>0 (248.54)
              #introduction<=0 (115.21) [4,0,0,0] course
              #introduction>0 (-2.00) [3,0,0,0] course
          #handouts>0 (579.19)
            #language<=0 (210.71)
              #group<=0 (59.77) [5,0,0,0] course
              #group>0 (-2.00) [2,0,0,0] course
            #language>0 (131.56) [7,0,0,0] course
#materials>0 (606.89)
```



```

#problem<=0 (331.51)
  #fall<=0 (37.23) [4,0,0,0] course
  #fall>0 (108.97) [5,0,0,0] course
#problem>0 (139.12) [7,0,0,0] course
-----
Number of clusters (leaves): 24
Correctly classified instances: 90 (75%)

```

With the default compression cutoff we obtain a very large tree, however the classes-to-clusters accuracy is substantially higher (75%). With a cutoff of 620 we can reduce the tree to 4 nodes.

```

> java MDLcluster cornell-binary200.arff 620

Attributes: 201
Ignored attribute: document_class
Instances: 119 (sparse)
Attribute-values in original data: 400
Numeric attributes with missing values (replaced with mean): 0
Minimum encoding length of data: 47046.82
-----
(776.71)
#materials<=0 (622.36)
  #handouts<=0 (663.86)
    #assignments<=0 (465.53) [7,34,20,21] faculty
    #assignments>0 (248.54) [7,0,0,0] course
  #handouts>0 (579.19) [14,0,0,0] course
#materials>0 (606.89) [16,0,0,0] course
-----
Number of clusters (leaves): 4
Correctly classified instances: 71 (59%)

```

The accuracy drops, but we obtain very good clusters for the “course” category. They are all “pure” (single class) and describe well course web pages with the presence of the terms “materials”, “handouts” and “assignments”.

6. Further experiments

- Select different number of attributes from the cornell data and find the number of attributes that maximizes classes-to-clusters accuracy. Explain why the attributes in the clustering tree change when the number of the selected attributes changes.
- Vary the compression cutoff to find good clusters for each of the four categories.
- Vary the compression cutoff to find a clustering that includes all four categories.
- Create TF and TFIDF representations of the cornell data and do the experiments described above with these data sets.
- Apply the MDL discretization algorithm (see the manual [2]) to the TF and TFIDF data sets and do the experiments described above.
- Create data sets from the web pages of other universities and do the experiments described above.

7. References

4. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam and S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web, in Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/>
5. Zdravko Markov, Java Classes for MDL-based Attribute Ranking and Clustering, <http://www.cs.ccsu.edu/~markov/MDLclustering/MDLmanual.pdf>
6. Zdravko Markov. MDL-based Unsupervised Attribute Ranking, Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS-26), St. Pete Beach, Florida, USA, May 22-24, 2013, AAAI Press 2013, pp. 444-449. <http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5845/6115>
7. Zdravko Markov and Daniel T. Larose. MDL-Based Model and Feature Evaluation, in Chapter 4 of *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Wiley, April 2007, ISBN: 978-0-471-66655-4.
8. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009). The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.