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

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.ccsu.edu/~markov/MDLclustering/. This site also provides a manual describing their functionality and usage [2].

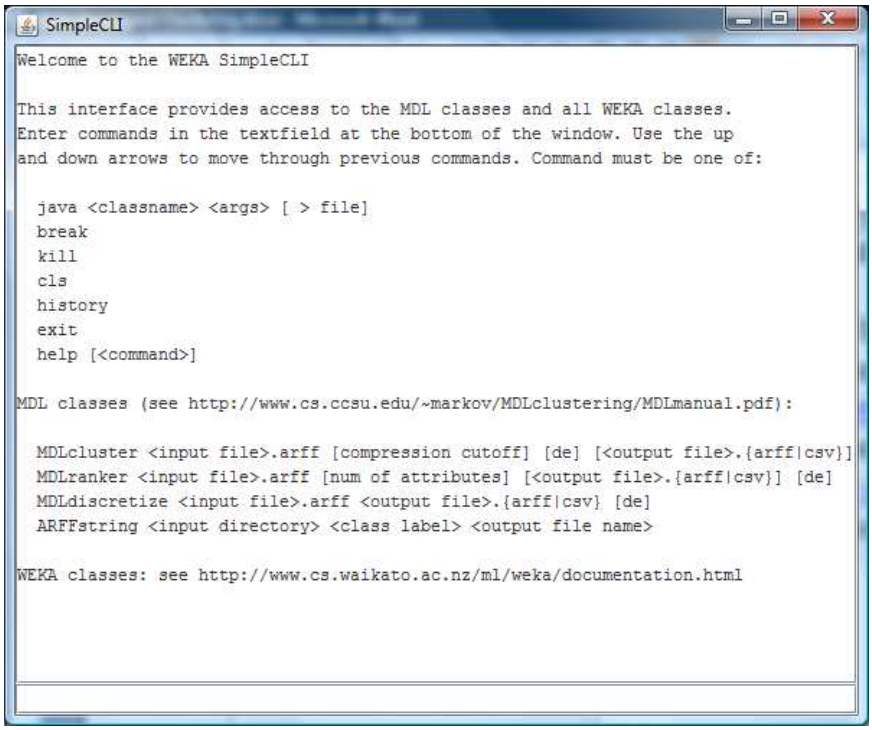
2. Data Collection

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.



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 and 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 represents 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 selected 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

1. 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/

2. Zdravko Markov, Java Classes for MDL-based Attribute Ranking and Clustering, http://www.cs.ccsu.edu/~markov/MDLclustering/MDLmanual.pdf

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. http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5845/6115

4. 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.

5. 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.