Data and feature extraction with MATLAB

Objective: To get acquainted with the way data can be represented in Matlab, how data can be imported from other data sources, and how data can be filtered and visualized using principal component analysis (PCA). Upon completing this exercise it is expected that you:

- Understand how data can be represented as vectors and matrices in Matlab
- Understand the *bag of words* representation for text documents including filtering methods based on removal of stop words and stemming.
- Can import data into Matlab from Excel and Matlab `.mat` files.
- Can apply and interpret principal component analysis (PCA) for data visualization.

Material: Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, *"Introduction to Data Mining"* 2.1-2.3 + (A) + B.1 as well as the files in the exercise 2 folder available from Campusnet.

Preparation: Exercise 1 MATLAB Help: MATLAB help is obtained by typing `helpdesk` at the MATLAB command prompt or by typing `help <function name>` in the command prompt. In practice, the fastest and easiest way to get help in Matlab is often to simply Google your problem. For instance: `"How to add legends to a plot in Matlab"` or the content of an error message. In the later case, it is often helpful to find the *simplest* script or input to script which will raise the error.

Software installation: Extract the Matlab toolbox from Campusnet. Start Matlab and go to the `<base-dir>/02450Toolbox_Matlab/` directory using the command `cd('<base-dir>/02450Toolbox_Matlab/')` and run `setup.m`. Consider storing your scripts for exercise 2 in: `<base-dir>/02450Toolbox_Matlab/week2/`. Representation of data in Matlab:

	Matlab var.	Type	Size	Description
	`X`	Numeric	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	Cell array	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	Numeric	Scalar	Number of data objects.
	`M`	Numeric	Scalar	Number of attributes.
Classification	`y`	Numeric	$N \times 1$	Class index: For each data object, `y` contains a class index, $\texttt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
Classification	`classNames`	Cell array	$C \times 1$	Class names: Name (string) for each of the C classes.
Classification	`C`	Numeric	Scalar	Number of classes.

2.1 The document-term matrix

An important area of research in machine learning and data mining is the analysis of text documents. Here, important tasks are to be able to search documents as well as group related documents together (clustering). In order to accomplish these tasks the text documents must be converted into a format suitable for data modeling. We will use the *bag of words* representation. Here, text documents are stored in a matrix $\mathbf{X}$ where x_{ij} indicate how many times word j occurred in document i.

Suppose that we have 5 text documents [4], each containing just a single sentence.

Document 1: The Google matrix P is a model of the internet.
Document 2: P_{ij} is nonzero if there is a link from webpage i to j.
Document 3: The Google matrix is used to rank all Web pages.
Document 4: The ranking is done by solving a matrix eigenvalue problem.
Document 5: England dropped out of the top 10 in the FIFA ranking.

2.1.1 Propose a suitable *bag of words* representation for these documents. You should choose approximately 10 key words in total defining the columns in the document-term matrix and the words are to be chosen such that each document at least contains 2 of your key words, i.e. the document-term matrix should have approximately 10 columns and each row of the matrix must at least contain 2 non-zero entries.

2.1.2 In practice, the above procedure is carried out automatically. We will use a toolbox called TMG [2] (included in the 02450 Toolbox). Use TMG to generate a document-term matrix and to convert it into the format described in the beginning of the exercise (Representation of data in Matlab).

Hints:

- *Type* `help tmg` *to learn how to use the TMG toolbox.*
- *The text documents are stored in the file* `Data/textDocs.txt`.
- *Sparse matrices can be converted to full (normal) matrices using the command* `full`.
- *Matrices can be transposed using the single quotation mark,* `'`.
- *The function* `cellstr` *can be used to create a cell array of strings from a character array.*
- *If you are stuck, take a look at solution in* `ex2_1_2.m`.

Compare the generated document-term matrix to the one you generated yourself.

Stop words are words that one can find in virtually any document. Therefore, the occurrence of such a word in a document does not distinguish the document from other documents. The following is the beginning of one particular stop word list:

> a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allov, ahnost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, around, as, aside,ask, ...

When forming the document-term it is common to remove these specified stop words.

2.1.3 The generated document-term matrix contains words that carry little information such as the word "the". We will remove these words as they can be interpreted as "noise" carrying no information about the content of the documents. Compute a new document-term matrix with stop words removed.

Hints:

- *Type* `help tmg` *to learn how to include a list of stop words in the TMG toolbox.*
- *A list of stop words is stored in the file* `Data/stopWords.txt`.
- *Notice the following options to TMG.*

```
TMG(FILENAME, OPTIONS) defines optional parameters:
- set OPTIONS.stoplist='filename.txt' to set the stoplist
  to use the file filename.txt.
  i.e. a list of common words that we don't use for
  the indexing (default no stoplist used).
```

- *If you are stuck, take a look at the solution in* `ex2_1_3.m`.

Inspect the document-term matrix: How does it compare to your original matrix?

Stemming denotes the process for reducing inflected (or sometimes derived) words to their stem, base or root form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Clearly, from the point of view of information retrieval, no information is lost in the following stemming reduction:

$$\left.\begin{array}{l} \textit{computable} \\ \textit{computing} \\ \textit{computed} \\ \textit{computational} \\ \textit{computation} \end{array}\right\} \rightarrow \textit{comput}$$

2.1.4 Document 3, 4 and 5 have the word "rank" in common. However in document 4 and 5 this word is stored as a the separate word entry "ranking" in the document-term matrix whereas in document 3 it is stored as the word entry "rank" . As such, the document-term matrix does not indicate that document 3, 4 and 5 share the word "rank". By the use of stemming we can obtain a matrix that indicate that the word "rank" appears in all 3 documents. Enable stemming in TMG and compute a new document-term matrix.

Hints:

- *Type* `help tmg` *to learn how to enable stemming in the TMG toolbox.*
- *Notice the following options to TMG.*
  ```
  TMG(FILENAME, OPTIONS) defines optional parameters:
  - OPTIONS.stemming: Indicates if the stemming algorithm
    is used (1) or not (0 - default).
  ```
- *If you are stuck, take a look at the solution in* `ex2_1_4.m`.

Inspect the document-term matrix: How does it compare to your original matrix? Can you get to the same result as shown below?

Based on our document-term representation we can now make simple searches (queries) in our documents based on some form of similarity measure between our query vector and document-term representation. Lets say we want to find all documents that are relevant to the query "**solving** for the **rank** of a **matrix**." This is represented by a query vector, $\boldsymbol{q}$, constructed in a way analogous to the document-term matrix, $\boldsymbol{X}$:

$$
\boldsymbol{X} = \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{drop} & \text{eigenvalu} & \text{england} & \text{fifa} & \text{googl} & \text{internet} & \text{link} & \text{matrix} & \text{model} & \text{nonzero} & \text{page} & \text{problem} & \text{rank} & \text{solv} & \text{top} & \text{web} & \text{webpag}
\end{array} \\
\left[\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}\right]
\end{array}
$$

$$
\boldsymbol{q} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}
$$

2.1.5 We will use the *cosine distance* as a measure of similarity between the i'th document $\boldsymbol{x}_i$ and the query vector $\boldsymbol{q}$, i.e. $\cos(\boldsymbol{q}, \boldsymbol{x}_i) = \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|} \cdot \frac{\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|} = \frac{\boldsymbol{q}\boldsymbol{x}_i^\top}{\|\boldsymbol{q}\|\|\boldsymbol{x}_i\|}$. (We will later in the course learn much more about measures of similarity). Compute the cosine similarity between each document and the query, and show that Document 4 is most similar to the query.

Hints:

- *You can extract a document (row of the* `X` *matrix) using the command* `x=X(i,:)` *where* `i` *is the index of the document.*
- *Dot products between two row vectors can be computed as using the function* `dot` *or as* `q*x'`
- *The norm of a vector can be computed using the function* `norm`.
- *If you are stuck, take a look at the solution in* `ex2_1_5.m`.

Explain what documents, according to our similarity measure, are most related to the query.

2.2 Importing data from Microsoft Excel

Many experimenters have a habit of using Microsoft Excel as their tool to record measurements from experiments. Fortunately Matlab can read Excel files, although it usually needs a bit of tweaking.

The data used in this exercise is based on data from a chemical sensor obtained from the NanoNose [1] project, see also [3]. The data contains 8 sensors named by the letters *A–H* measuring different levels of concentration of Water, Ethanol, Acetone, Heptane and Pentanol injected into a small gas chamber. The data will be represented in matrix form such that each row contains the 8 sensors measurements (i.e. sensor A-H) of the various compounds injected into the gas chamber.

2.2.1 Inspect the file `<base-dir>/02450Toolbox_Matlab/Data/nanonose.xls` and make sure you understand how the data is stored in Excel.

Load the data into Matlab using the `xlsread` function, and get it into the standard data matrix form as described in the beginning of this document.

Hints:

- *Type* `help xlsread` *to learn how to use the function for reading Excel files into Matlab.*
- *Note that by default* `xlsread` *only reads numeric cells.*
- *Note that using the syntax*
  `[NUMERIC,TXT,RAW]=xlsread(FILE)`
  you can get both the numeric data, text data, and raw data as a cell array.
- *You can use indexing to get submatrices of the returned matrices, e.g.,* `X = NUMERIC(:,3:10)` *to get columns 3–10 and all rows.*
- *If you are stuck, take a look at the solution in* `ex2_2_1.m`.

There are 90 data objects with 8 attributes each. Can you get the correct data matrix X of size 90×8?

Look at the solution in `ex2_2_1.m` to see how the attribute names and class names and indices can be extracted. Run the script and look at the variable that it returns.

2.2.2 The data resides in an 8 dimensional space where each dimension corresponds to each of the 8 NanoNose sensors. This makes visualization of the raw data difficult, because it is difficult to plot data in more than 2–3 dimensions.

Plot the two attributes A and B against each other in a scatter plot.

Hints:

- *Type* `doc plot` *to learn how the Matlab plot function works.*
- *The attributes A and B are the first and second columns of the matrix* $\boldsymbol{X}$*.'*
- *You can use indexing to get the columns out of the matrix, e.g.,* `x=X(:,1); y = X(:,2);`
- *Notice that the third argument of the* `plot` *command can be used to set a plot symbol. For example, the command* `plot(x,y,'o')` *plots a scatter plot with circles.*
- *If you are stuck, take a look at the solution in* `ex2_2_2.m`.

Try to change the dimensions that are plotted against each other.

We will use principal component analysis to reduce the dimensionality of the data. PCA is computed by subtracting the mean of the data, $\boldsymbol{Y} = \boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}$ (where $\boldsymbol{\mu}$ is a (row) vector containing the mean value of each attribute and $\mathbf{1}$ is a N by 1 column vector of ones in all entries) and then calculating the singular value decomposition (SVD) of the zero mean data, i.e. $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$.

From PCA we can find out how much of the variation in the data each PCA component accounts for. This is given by

$$\rho_m = 1 - \frac{\|\boldsymbol{Y} - \boldsymbol{u}_m s_{mm} \boldsymbol{v}_m^\top\|_F^2}{\|\boldsymbol{Y}\|_F^2} = \frac{s_{mm}^2}{\sum_{m=1}^{M} s_{m,m}^2},$$

i.e. the squared singular value of the given component divided by the sum of all the squared singular values.

2.2.3 Compute the PCA of the NanoNose data and plot the percent of variance explained by the principal components.

Hints:

- *You can use the function* `mean` *to compute the mean of the data.*
- *You should compute the mean for each attribute, i.e., the vector of means should have M elements.*
- *You cannot directly subtract a vector from a matrix. One way to accomplish this is through the* `bsxfun` *function. Try typing* `help bsxfun`.
- *Notice the example:*
```
Subtract the column means from the matrix A:
A = bsxfun(@minus, A, mean(A));
```
- *Try the command* `Y = bsxfun(@minus, X, mean(X));`
- *You can use the function* `svd` *to compute the SVD.*
- *To extract the diagonal from a matrix, use the command* `diag`.
- *If you are stuck, take a look at the solution in* `ex2_2_3.m`.

Can you verify that more than 90% of the variation in the data is explained by the first 3 principal components?

2.2.4 Plot principal component 1 and 2 against each other in a scatterplot.

Hints:

- *Data can be projected onto the principal components using* `Z = Y*V` *or equivalently computed directly from the SVD as* `Z = U*S`.
- *You learned how to make a scatter plot in Exercise 2.2.2.*
- *If you are stuck, take a look at the solution in* `ex2_2_4.m`.

What are the benefits of visualizing the data by the projection given by PCA over plotting two of the original data dimensions against each other? Compare with the scatter plots of attributes you made in Exercise 2.2.2.

2.2.5 Which of the original attributes does the second principal component mainly capture the variation of and what would cause an observation to have a large negative/positive projection onto the second principal component?

Hints:

- *The columns of* `V` *gives you the principal component directions*
- *The data is projected onto the second principal component by* `Y*V(:,2)`

2.3 Structure in handwritten digits

The US Postal Service (USPS) wanted to automate the process of sorting letters based on their zip-codes. We will presently consider a dataset of USPS handwritten digits available at `http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html`, see also [5].

2.3.1 Load zipdata.mat by typing `load Data/zipdata.mat`. There are two datasets containing handwritten digits `testdata` and `traindata`.

2.3.2 Inspect and run the script `ex2_3_1.m` to visualize the first digit of the traindata (the script uses the function reshape to turn a digit vector into an image and `image` to display the image).

2.3.3 Inspect and run the script `ex2_3_2.m`. Show that it requires 22 PCA components to account for more than 90% of the variance in the data. Show that the first principal component is almost sufficient to separate zeros and ones. Examine the first principal component and discuss and understand what it captures.

2.3.4 Change the value of `K` and show that reconstruction accuracy improves when more principal components are used. How many principal components do you need to be able to see the different digits properly? What happens if you set `K=256`?

2.3.5 Try decomposing one digit at a time. Hint: Modify the variable `n` to contain only a single digit. Explain what happens to the principal components when only a single digit type is analyzed compared to when all digit types are analyzed at the same time.

2.4 Extra challenge

We will later in the course learn various methods for classification. Among the approaches we will learn is K-nearest neighbor (KNN) classification. For now we will consider the KNN classifier a black box that we will use to evaluate how well we can determine the digit class in the space given by the K first principal components, i.e. after filtering out the PCA components with smallest singular values which we consider components pertaining to noise.

2.4.1 Inspect and run the script `ex2_4_1.m` and see how well we are able to classify the digits when we use say `K=10` PCA components, `K=40` PCA components and the whole data, i.e `K=256` PCA components. Show that the classifier is best when using around 40–60 PCA components, and explain why that is so.

2.5 Tasks for the report

After today's exercise you should be able to load your data into Python (from Excel and from Matlab) and put it in the format described in the section on "Representation of data in Python" at the beginning of today's exercise. You should also be able to explain what types of attributes are in your data (i.e. discrete/continous, Nominal/Ordinal/Interval/Ratio, see also today's lecture) as well as be able to apply and interpret the results of a principal component analysis (PCA) of your data. Notice, that there are three aspects that needs to be described in the PCA analysis for the report:

- The amount of variation explained as a function of the number of PCA components included,
- the principal directions of the considered PCA components,
- the data projected onto the considered principal components.

References

[1] Nanonose project.

[2] Tmg software.

[3] Tommy S Alstrøm, Jan Larsen, Claus H Nielsen, and Niels B Larsen. Data-driven modeling of nano-nose gas sensor arrays. In *SPIE Defense, Security, and Sensing*, pages 76970U–76970U. International Society for Optics and Photonics, 2010.

[4] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007.

[5] Jonathan J. Hull. A database for handwritten text recognition research. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(5):550–554, 1994.