SCC361 Artificial Intelligence

Lab Session 2 Feature Extraction

Key aims of the Session:

- Loading and preparing text and numeric data
- Data Visualisation
- Creating and Visualising Bag of Word Models
- Basic Feature Processing Techniques

Contents

Introduction

This lab is split into two exercises. The first is to create and use a Bag of Words (BoW) approach on various types of text. The second involves initial data feature processing with general data.

Editor vs Command Window

While you can complete this worksheet by entering commands sequentially into the command window, you may find it more useful to save and run the commands as a script. To open up the script editor, type `edit` into the command window. Don't forget to save your script so that you don't lose it. Once saved, you can run your script by clicking "Run" on the top ribbon or by typing the script's name into the command window. Please note that MATLAB script filenames should have a .m extension.

Exercise 1: Natural Language Processing: Bag of Words (BoW)

In this exercise, we will carry out some examples of creating bag of words (BoW) from text data. A bag of words, in its simplest form, is a count of each word per document in a given text collection's vocabulary (the list of words that appear in the text collection).

The standard process that we'll use involves the following steps:

1. Prepare the data
2. Build a bag of words
3. Find the most common words in the bag
4. Remove any stopwords
5. Calculate the Tf-idf Matrix
6. Produce a word cloud

These steps will be described in more detail in the below examples.

1.1: Artificial Example 1: Words with given number of occurences

In this example, we'll create our own example of a collection of text with numbers of instances manually inputted, build a BoW model, and display the results.

Prepare the Data

Note: the array and matrix below are formatted to be copied and pasted into matlab.

Create an array of *unique* words:

```
uniqueWords = ["a" "an" "another" "example" ...

"final" "sentence" "third"];
```

Add a matrix of the amount of times each word appears in a sentence:

```
counts = [ ...
1 2 0 1 0 1 0;
0 0 3 1 0 4 0;
1 0 0 5 0 3 1;
1 0 0 1 7 0 0];
```

Add up the total for each column to find the total number of instances of each word.

```
sum(counts)
```

- Which words are the most common?
- Why does the array of words have to contain unique words?

Bag of Words

Build the bag of words:

```
bag = bagOfWords(uniqueWords,counts);
```

Most Common Words

Display the top 7 entries in the bag:

```
topkwords(bag,7)
```

Stopwords

Stopwords (frequently used function words, such as “a”, “the”, “and”, etc.) tend to skew the counts in the bag of word model. We can remove them from a model by inputting a list of stopwords to the `removeWords` function. Matlab has a built-in set of stopwords in the variable `stopWords`.

Remove any stopwords:

```
newBag = removeWords(bag,stopWords);
```

TF-IDF Matrix

TF-IDF (Term Frequency–Inverse Document Frequency)[1] is widely used in information retrieval and text analysis. It helps to reduce the impact of words which appear frequently across a collection of documents (i.e. they do not distinguish individual documents). TF-IDF increases with the number of occurrences within a document (term frequency), but also increases with the rarity of the term in the collection as a whole (inverse document frequency). Here, a ‘document’ is a row in our counts matrix or BoW.

Calculate the TF-IDF matrix for the two BoWs and consider the differences between them:

```
M1 = tfidf(bag);
full(M1)
M2 = tfidf(newBag);
full(M2)
```

The `full` function is used to convert a sparse matrix (outputted from `tfidf`) of nonzero elements, to a full matrix including zeroes.

Word Cloud

Produce word clouds and note the differences:

```
figure(1)

subplot(1,2,1)
wordcloud(bag);
title('Wordcloud')
subplot(1,2,2)
wordcloud(newBag);
title('Refined Wordcloud')
```

- Try varying the words and instances to produce new clouds.

[1] https://en.wikipedia.org/wiki/Tf–idf

1.2: Artificial Example 2: Creating a BoW from sentences

In this example, we'll create a bag of words from a small set of sentences. In this case, we won't specify a unique set of words or numbers of instances; instead, we'll get Matlab to do this for us.

Prepare the Data

Create the sentences that we want to use (*formatted to be copied and pasted into matlab)*:

```
strs = [...
"an example of a short sentence" ...
"a second short sentence" ...
"and this is a slightly longer sentence than the short sentences"];
```

Tokenize (split into 'words' or tokens) the sentences to prepare them for the BoW:

```
documents = tokenizedDocument(strs);
```

Bag of Words

We call the Matlab built-in function `bagOfWords()` to create the bag of words:

```
bag = bagOfWords(documents);
```

Analysis

Now follow the steps from the previous example to (i) find out the most common words from the bag, (ii) remove stopwords, (iii) print the TF-IDF matrix, (iv) display the word clouds.

- How many stopwords did this document contain?

1.3: Example 3: Creating a BoW from a Document

Let's now consider an example text data file "stephenfry.txt", which contains a series of old Tweets from Stephen Fry, one tweet per line.

Prepare the Data

We first need to extract the text from the file, split the text into documents at newline characters, and then tokenize the documents.

Extract the data from the file:

```
str = extractFileText("stephenfry.txt");
```

Split the data on newline:

```
textData = split(str,newline);
```

Tokenize the strings in textData

```
documents = tokenizedDocument(textData);
```

With `whos documents`, you can note that there are 2779 documents (documents here refers to each line, i.e. each Tweet).

`length(documents.Vocabulary)` tells us there are 12067 unique words, our vocabulary size.

Bag of Words and Analysis

Now follow the steps from the previous examples to (i) build the bag of words (ii) find out the most common 10 words from the bag, (iii) remove stop words, (iv) print the first 10 rows and columns of the TD-IDF matrix, (v) display the word clouds.

- You may notice that there is a lot of punctuation appearing in the most frequent words and word clouds. Try removing it by supplementing the `stopWords` variable with additional entries. You will need to inspect the `stopWords` variable to check the correct formatting. Reproduce the word cloud to view the impact.

Extra optional tasks:

- You may also notice that even when stopwords are removed, stopwords with an initial capital (e.g. "The") still appear. There are two approaches to try to fixing this behaviour. The `removeWords` function can be told not to be case-sensitive by adding the option `'IgnoreCase', true`. Or the whole text can be made lowercase in the initial processing with the `lower` function. This is often done in NLP tasks to avoid counts being split between upper and lower case. Fix the casing behaviour, and view the results.
- The default MATLAB tokenizer does quite a good job at recognising tokens in social media text, like hashtags, URLs, and mentions. There are some tokens that may need processing differently. Some ways of dealing with these are to extend further the list of words to remove, give instructions to improve the tokenizer, and to replace strings in the whole text. A particular issue to resolve is "&amp;" being present in the text. These can be replaced in the whole text with the `replace` function, or tokenized separately using the `'CustomTokens'` option in `tokenizedDocument`, and then removed by extending stopWords. Deal with "&amp;" and other issues to improve the quality of the word clouds and frequency lists.

1.4: Example 4: Creating a BoW from a set of documents

Let's now consider creating a bag of words from a set of documents. In this case, we'll import the data into MATLAB using a file datastore. For this example, we'll use the files in the zip file "Shakespeare.zip", which contains 120 text files of Shakespeare plays.

Prepare the Data and build the Bag of Words

Extract the Shakespeare.zip contents to a folder called "Shakespeare"

```
unzip Shakespeare.zip Shakespeare
```

Create a file datastore for the example text files. All text files are ".txt" files, so we can use the wildcard * to retrieve all. Specify the read function to be extractFileText and build the datastore:

```
readFcn = @extractFileText;
fds = fileDatastore('Shakespeare/*.txt','ReadFcn',readFcn);
```

Since we have multiple files from which we will need to extract data, we will create an empty bag of words model to act as initialisation. Then, at each stage of the loop, we can add data to it.

```
bag = bagOfWords;
```

Loop over the files in the datastore and read each file. Tokenize the text in each file and add the document to the bag:

```
while hasdata(fds)
    str = read(fds);
    document = tokenizedDocument(str);
    bag = addDocument(bag,document);
end
```

View the final bag-of-words model:

```
bag
```

- How many words are contained in the vocabulary?

Analysis

Now that we have the bag of words, follow the steps from the previous example to (i) find out the most common 10 words from the bag, (ii) remove stopwords, (iii) print the first 10 rows and columns of the TF-IDF matrix, (iv) display the word clouds.

Improve the tokenization and resulting word frequencies and word clouds by making appropriate changes to the text preprocessing, as in Example 3.

Exercise 2: Initial data feature processing

For this question, we will carry out some initial analysis and pre-processing on some example data. This data is contained in the zip folder "data_files.zip". This includes two .csv files: (i) "iris_data_150samples.csv" and (ii) "iris_missing_values.csv".

2.1: Load and Prepare the data

Extract the files from the .zip file:

```
unzip data_files.zip data_files
```

The file "iris_data_150samples.csv" contains 6 columns: 4 for the measurements and 2 for class labels in numerical and string formats.

Use the `readtable()` function to read the data into MATLAB, into a variable `ID`.

Check the size of the table and print every 10th row.

Find the list of distinct classes of data in the "Class2" column:

```
unique(ID.Class2)
```

Based on our review of the data above, we can see that for each entry, there are four measures (Measure1,… Measure4), and two classes (Class1 and Class2), which we know are equivalent. We'll consider the measures to be features and the classes to be labels.

Extract the features and labels from the table, store them in new variables, check their size and type:

```
features = table2array(ID(:,1:4));
size(features)
labels = table2array(ID(:,6));
size(labels)
whos features labels
```

Note that, had we not used the `table2array` function, the “features” and “labels” would have been exported as tables.

2.2: Visualising the Clustering of the Data

We can plot a 2D visualisation of column pairs of the data using the `gscatter` function. Let’s look at the first and second columns of the features i.e. Measures 1 and 2:

```
figure(1)
gscatter(features(:,1), features(:,2));
title('Scatter plot: Iris Dataset');
xlabel('Measure 1');
ylabel('Measure 2');
```

Can you observe any properties of the data? It is not easy to see a pattern and impossible to consider patterns relating to the classes unless we include the class (label) information. Fortunately, we can add this quite easily:

```
figure(2)
gscatter(features(:,1), features(:,2), labels,'rgb');
title('Scatter plot: Iris Dataset');
xlabel('Measure 1');
ylabel('Measure 2');
```

We can now observe that the “setosa” class is clustered together and away from the other classes when the first two Measures are used.

- Try plotting other combinations of the columns to see if you will gain some more insight from the data.

2.3: Handling Missing Values

Sometimes, datasets contain missing values (entries that are not entered/captured wrongly). In MATLAB, the standard representation for missing values is NaN for double, single, duration, and calendarDuration. Other formats are NaT, <missing>, <undefined>, ‘ ’, {‘ ’} for datetime, string, categorical, char and cell of character vectors respectively. The `ismissing` function can help you to identify missing elements in an array or table. The `find` function can be used to return nonzero indicies.

- Load the “iris_misssing_values.csv” into a variable `ID` and inspect the data for missing elements.
- How many missing elements are there in the data?
- Which measures have missing values?
- Which entries have missing values?

Where there are missing values in data, two possible ways of handling this are:

1. removing missing values
2. filling missing values

Removing missing values

We can use the function `rmmissing()` to remove entries with missing values:

```
rem_rows = rmmissing(ID);
```

This will remove all rows that have missing values. To remove columns instead of rows, we can specify:

```
rem_cols = rmmissing(ID,2);
```

NB: this can delete valuable information by removing entire rows and columns because of few (or single) missing values.

Alternatively, we can specify a limit for the number of missing elements for any rows or columns to be removed:

```
rem_rows_2 = rmmissing(ID, 'MinNumMissing', 2);
rem_cols_2 = rmmissing(ID, 2, 'MinNumMissing', 2);
```

This will remove rows (resp. columns) with 2 or more missing elements in their row (resp. column) entries.

- How many rows and columns remain after each of the above operations?

Filling missing values

To avoid deleting data, you can instead fill the missing elements. A common approach is to replace it with the mean value of the feature. For example:

```
mFeatures = mean(features, 'omitnan');
```

```
featuresfilled = fillmissing(features,'constant', mFeatures);
```

- Can you fill in all missing elements in the data with the median value of the feature?

2.4: Handling Outliers

Outliers can skew or mislead models into producing wrong results and handling outliers is an important pre-processing step. The function `isoutlier` can help to identify these. By default, MATLAB uses the median value to detect an outlier, i.e. any value that is 3 standard deviation away from the median.

- Try to use the `isoutlier` function to locate outliers in your data.
- Try to detect outliers using the mean.
- Check the difference between the mean and the median.

2.5: Classification

Add the following samples to the graphs created in Section 2.2 and use this to estimate the class that they belong to.

	Measure 1	Measure 2	Measure 3	Measure 4
Sample 1	6.3	2.6	4.1	1.2
Sample 2	4.7	3.5	1.5	0.3
Sample 3	7.1	2.9	5.5	2.1