AARHUS UNIVERSITY

Exam Portfolio for Language Analytics

Link to GitHub repository: https://github.com/bokajgd/Language-Analytics-Exam

May 27th 2021

Jakob Grøhn Damgaard, 201808996
CDS Tilvalg: Language Analytics – Teacher: Ross Deans Kristensen-McLachlan.

Table of Contents

Introduction

This document constitutes my collection of assignments from Spring 2021 course *Language Analytics*, part of the bachelor's elective in Cultural Data Science. These assignments all revolve around conducting computational analysis of language data with varying purposes in Python (Van Rossum & Drake, 2009). Four assignments were assigned by the teacher throughout the semester while the last one comprises a self-assignment project. All source scripts for each assignment are coded according to principles of object-oriented programming. Hence, each script consists of a class that houses different functions and is initialized when the script is executed from a command line. In this document, each assignment has appertaining sections that provide a short description of the topic and tasks, a section on how to interact with and run the code, a section describing the methods used and, lastly, a short reflection on the end results.

General Instructions

Link to overall repository: https://github.com/bokajgd/Language-Analytics-Exam

This section provides a detailed guide for locally downloading the code from GitHub, initialising a virtual Python environment, and installing the necessary requirements. In order to maximise user-friendliness and ease the processes mentioned above, all projects have been collated into one single GitHub repository. Therefore, all code can be fetched into one local folder and as there are no overlapping dependencies, only a single virtual environment is needed. Please note, a local installation of Python 3.8.5 or higher is necessary to run the scripts.
To locally download the code, please open a terminal window, redirect the directory to the desired location on your machine and clone the repository using the following command:

```
git clone https://github.com/bokajgd/Language-Analytics-Exam
```

Then, proceed to execute the Bash script provided in the repository for initialising a suitable virtual environment:

```
./create_venv.sh
```

This command may take a few minutes to finalise since multiple packages and libraries must be collected and updated. When it has run, your folder should have the following structure (folder depth of 2):

```
.
├── A2-Collocation
│   ├── README.md
│   ├── data
│   ├── output
│   ├── src
│   └── viz
├── A3-Sentiment-Analysis
│   ├── README.md
│   ├── data
│   ├── output
│   └── src
├── A4-Network-Analysis
│   ├── README.md
│   ├── data
│   ├── output
│   ├── src
│   └── viz
├── A5-Supervised-Learning
│   ├── README.md
│   ├── data
│   ├── output
│   ├── src
│   └── viz
├── Language_Analytics_Exam.pdf
├── README.md
├── SA-LDA-Feature-Extraction
│   ├── README.md
│   ├── data
│   ├── output
│   ├── src
│   └── viz
├── create_venv.sh
└── requirements.txt
```

You can verify this structure by running the following command:

```
tree -I lang_analytics_venv -L 2
```

If everything checks out, you should be ready to execute the code scripts located in the respective assignment folders. This is the base folder from which you are advised to navigate back to whenever you are finished running and assessing the scripts in one of the assignment folders. Always remember to activate the virtual environment before executing scripts in the terminal command line. This command can only be executed when you are in the main folder.

```
source lang_analytics_venv/bin/activate
```

The same goes for deactivating the environment when use is ceases:

```
deactivate
```

All code has been tested in Python 3.8.5 on a 2020 MacBook Pro 13'', 2 GHz Quad-Core Intel Core i5, 16 GB Ram running macOS Big Sur (11.2.6). Thus, following the instructions should allow for smooth execution on MacOS and Linux machines. If you are running the code in any other operating system, please adapt commands to that syntax.

More detailed instructions regarding the execution of the individual scripts can be found in the sections on the respective assignments.

A2 – Collocation: Quantifying the Attraction Between Words

Link to the assignment folder (embedded in the overall repository) can be found here: https://github.com/bokajgd/Language-Analytics-Exam/tree/main/A2-Collocation

Description

"You shall know a word by the company it keeps!" (Firth, 1957). These renowned words (paraphrased) were written over 60 years ago but still form the foundation of the study of distributional semantics and language structure in general. Today, modern state-of-the-art machine learning models still represent the semantics of a word by analysing the context words that surround it (Wolf et al., 2018). A common, and more simple, approach from corpus linguistics for analysing and quantifying inter-word relationships is to look at *word collocation*. From a statistical point-of-view, two words are referred to as collocates, when they regularly appear closely together (within a designated span of words from each other) to such an extent that their co-occurrence frequency is statistically significant in some way (Baker, 2006). One metric for quantifying collocation association strength is to a calculate a *Mutual Information score* (MI score) (see information in *methods* section). Due to the non-random nature of language, most collocations are viewed as significant, and association measures such as *Mutual Information* are merely used to rank word relationships in magnitude.

For this assignment, we were asked to create a script for calculating word collocations for a given keyword to practice our Python string processing skills. More specifically, we were asked to produce a script capable of the following:

- Take a text corpus, keyword and a window size (number of words) as input parameters
- Find out how often each word co-occurs with the given keyword across the corpus and use this to calculate a MI score between the keyword word and all collocates across the corpus
- Generate and output a *.csv* file consisting of three columns: *collocate, raw_frequency, MI*

For the assignment, I have used a corpus of 100 English novels. This data is already in the folder (see *Usage*) but can also be downloaded from this link.

Usage

If not already open, open a terminal window and redirect to the home folder of the cloned repository (see *General Instruction*). Remember to activate the virtual environment. Then, jump into the folder called *A2-Collocation* using the following command:

```
cd A2-Collocation
```

Now, it should be possible to run the following command in to get an understanding of how the script is executed and which arguments should be provided:

```
# Add -h to view how which arguments should be passed
python3 src/A2-Collocation.py -h
usage: A2-Collocation.py [-h] [-key --keyword] [-ws --window_size]

[INFO] Calculating Word Collocation Mutual Information Scores

optional arguments:
  -h, --help            show this help message and exit
  -key --keyword        [DESCRIPTION] The name of the desired keyword
                        [TYPE]        str
                        [DEFAULT]     cat
                        [EXAMPLE]     -key cat
  -ws --window_size     [DESCRIPTION] How many words the sliding window
                                      should extend on both sides of the keyword
                        [TYPE]        int
                        [DEFAULT]     2
                        [EXAMPLE]     -ws 2
  -dd --data_directory  [DESCRIPTION] Path to directory where data is located
                        [TYPE]        str
                        [DEFAULT]     A2-Collocation/data/100_english_novels
                        [EXAMPLE]     -dd A2-Collocation/data/100_english_novels
```

The script can now be executed with the same output using either of the following commands:

```
# No arguments passed - the script reverts default values
python3 src/A2-Collocation.py

# With command line arguments given - same as default values
python3 src/A2-Collocation.py -key cat -ws 2
```

Note that execution may take a few minutes to complete.

I've simplified the command line arguments as much as possible to increase user-friendliness and, thus, no arguments are strictly required for the user to input in order to initiate a test run of the script. The data directory defaults to the directory of the provided corpus of English novels. Do, however, feel free to play around with the script by changing the keyword or window size. The outputted data frames are given a unique named based on these inputs.

Structure

The structure of the assignment folder can be viewed using the following command:

```
tree -L 2
```

This should yield the following graph:

```
.
├── README.md
├── data
│   ├── 100_english_novels
│   └── README.md
├── output
│   └── cat_collocates_ws_2.csv
├── src
│   └── A2-Collocation.py
└── viz
    └── collocation_example.png
```

The following table explains the directory structure in more detail:

Folder	Description
`data`	A folder containing the provided corpus of English novels in a *.txt* format. The files are in the *100_english_novels* subfolder. The *README.md* file contains a brief description of the dataset.
`src`	A folder containing the source code (*A2-Collocation.py*) created to solve the assignment.
`output`	A folder containing the output produced by the Python script. The script yields a *.csv* file with the file name ***<keyword>**_collocates_**<window size>**_ws.csv*
`viz`	A folder containing the *.png* visualisations for the README.md file

Methods

As stated in the *Introduction*, the script is coded using the principles of object-oriented programming. The main class of the script includes an **__init__** method which holds the set of statements used for solving the desired tasks. This collection of statements is executed when the class object is created. Furthermore, the class holds a series of functions created for solving the requested tasks which are called when needed in the **__init__.** The class object is created - and thus, the tasks are performed - whenever the main function is executed. This happens every time the module is executed as a command to the Python interpreter.

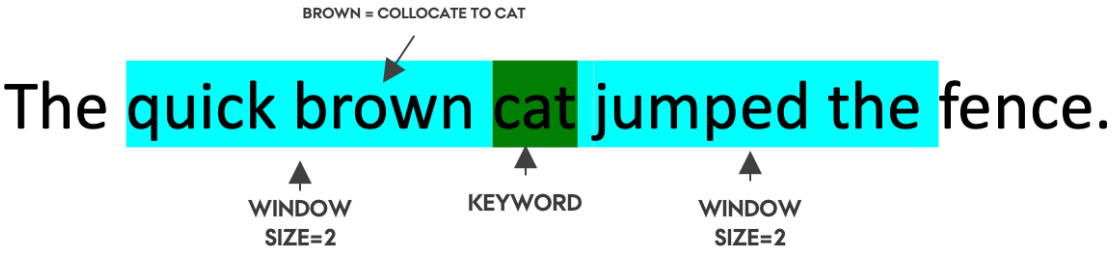

Figure 1 - Example of a window of size 2 around a keyword. All words highlighted in turquoise are collocates to the keyword 'cat'

Calculating the mutual information between a keyword and a collocate is reliant on several values in the following contingency table (Evert, 2004):

Table 1 - Contingency Table for Collocation

	$* \mid collocate$	$* \mid \neg\, collocate$	
$keyword \mid *$	O_{11}	O_{12}	$= R_1$
$\neg\, keyword \mid *$	O_{21}	O_{22}	
	$= C_1$		$= N$

The relevant numbers from *table 1* can more simply be explained as representing the following:

- O11 = Number of times the specific collocate appears within the given window of a keyword for entire corpus
- O21 = Number of times the specific collocate appears outside of the given window across entire corpus
- O12 = Number of other words except the specific collocate that appear in a given window near the keyword across a text/corpus
- R1 = Total number of all possible collocation pairs with keyword
- C1 = Total occurrences of the specific collocate either inside or outside of the context window
- N = Total word size of corpus

To calculate the MI, one must calculate the *expected collocation frequency, E11*:

$$E_{11} = \frac{R_1 * C_1}{N}$$

Lastly, the formula for MI is given by:

$$MI = log \frac{O_{11}}{E_{11}}$$

The script loads in the *.txt* files one-by-one, tokenizes them and appends them into one large list of tokenized words before calculating the relevant scores for finding an MI score and a raw collocate frequency. The general script is heavily reliant on the data handling package *Pandas* (McKinney et al., 2010) and *Numpy* (Harris et al., 2020) for array and matrix processing.

Discussion of Results

The script works as intended and matches all the requirements. *Table 2* shows an excerpt from the *.csv* file generated when running the script with its default settings.

cat_collocates_ws_2

	collocate	raw_frequency	MI
432	filippina	1	13.241493212324903
256	pussy	9	13.07156821088259
393	elaborates	2	12.241493212324903
61	spotty	4	11.241493212324903
189	plumy	4	11.241493212324903
108	shrewish	5	10.919565117437541

Table 2

In its current form, the script is, however, unable to handle one minor issue. As the individual texts are currently appended into one long list, the window occasionally laps across two different novels, which is obviously not ideal. Additionally, the efficiency of the code could potentially be optimised to minimise run time.

A3 – Sentiment Analysis

Link to the assignment folder (embedded in the overall repository) can be found here: https://github.com/bokajgd/Language-Analytics-Exam/tree/main/A3-Sentiment-Analysis

Description

For this assignment, we were tasked with performing dictionary-based sentiment analysis. Dictionary-based sentiment analysis is one of the simplest computational approaches to sentiment analysis and relies on cross-referencing the words in a text with a human-rated sentiment dictionary for calculating sentiment scores. NLP based sentiment analysis aims to identify and quantify the sentiment (often either positive, negative or neutral but can be extended to various moods) carried in a text (Feldmann, 2013).

For the analysis, we were provided with a data set of one million news headlines from Australian news media *ABC News* collected between 2003 and 2020. This data is already in folder (see *Usage*) but can also be found and downloaded from here. The assignment required us to create and save a plot of sentiment over time with a 1-month rolling average and a plot of sentiment over time with a 1-month rolling average with clear labels, legends etc.. Furthermore, a paragraph containing a brief discussion of the resulting trends should be phrased. In order to increase the quality of the plots, I decided to overlay a 1-year polarity average line plot on both plots to highlight the broader developments.

Usage

If not already open, open a terminal window and redirect to the home folder of the cloned repository (see *General Instruction*). Remember to activate the virtual environment. Then, jump into the folder called *A3-Sentiment-Analysis* using the following command:

```
cd A3-Sentiment-Analysis
```

Now, it should be possible to run the following command in to get an understanding of how the script is executed and which arguments should be provided:

```
python3 src/A3-Sentiment-Analysis.py -h
usage: A3-Sentiment-Analysis.py [-h] [-fn --filename] [-dd --data_directory]

[INFO] Sentiment Analysis on a Million Headlines

optional arguments:
  -h, --help            show this help message and exit
  -fn --filename        [DESCRIPTION] The name of the input data file
                        [TYPE]        str
                        [DEFAULT]     abcnews-date-text_subset.csv
                        [EXAMPLE]     -fn abcnews-date-text_subset.csv
  -dd --data_directory  [DESCRIPTION] Path to directory where data is located
                        [TYPE]        str
                        [DEFAULT]     A3-Sentiment-Analysis/data
                        [EXAMPLE]     -dd A3-Sentiment-Analysis/data
```

The script can now be executed with the same output using either of the following commands:

```
# No arguments passed - the script reverts default values
python3 src/ A3-Sentiment-Analysis

# With command line arguments given - same as default values
python3 src/ A3-Sentiment-Analysis -fn abcnews-date-text_subset_subset.csv
```

As executing the script on the full one million headlines is rather time consuming, I have included a subset of only the first 200.000 headlines in the *data* folder. This is the file that the script sets as its default input file. If you instead wish to run the script on the entire corpus (and generate the plots shown in the *results* section), feel free to execute the following command:

```
python3 src/ A3-Sentiment-Analysis -fn abcnews-date-text.csv
```

Structure

The structure of the assignment folder can be viewed using the following command:

```
tree -L 2
```

This should yield the following graph:

```
.
├── README.md
├── data
│   ├── abcnews-date-text.csv
│   └── abcnews-date-text_subset.csv
├── output
│   ├── Monthly_Rolling_Average_Polarity.png
│   └── Weekly_Rolling_Average_Polarity.png
└── src
    └── A3-Sentiment-Analysis.py
```

The following table explains the directory structure in more detail:

Folder	Description
`data`	A folder containing the raw data used for the assignment. This folder includes the full data set *abcnews-date-text.csv* and and a subset called *abcnews-date-text_subset.csv* which contains the first 20000 headlines.
`src`	A folder containing the source code (*A3-Sentiment-Analysis.py*) created to solve the assignment.
`output`	A folder containing the output produced by the Python script. The script yields two time series plots showing the development in mean headline polarity: • *Weekly_Rolling_Average_Polarity.png*: Shows the weekly rolling average • *Monthly_Rolling_Average_Polarity.png*: Shows the monthly rolling average

Methods

Akin to the script in assignment 2, the script is coded using the principles of object-oriented programming. See the first paragraph of the *A2-Collocation* methods section for a quick outline of the general script architecture.

The script employs the pipeline component *spaCyTextBlob* from the *spaCy* (Honnibal & Montani, 2017) library built on *TextBlob* (Loria, 2018) for the sentiment analysis. After calculating polarity scores for the individual headlines, polarity is grouped and averaged for every day and every year, respectively. Rolling averages are then calculated over a 7 and a 30 day interval and used for drawing two *Matplotlib* (Hunter, 2007) time series plots. These are, lastly, saved to the *output* folder.

Discussion of Results

It is difficult to draw any major conclusions or inferences, when eyeballing the plots displaying the rolling weekly and monthly averages in headline polarity scores (see *figure 2*). However, the plots display a slight dip in average scores during the first decaded which is subsequently followed up by a steady increase starting around 2010. It could be speculated that this rise may be a result of a surge in use of social media. The prevalence of these platforms may have pushed news media towards the use of more sensational headlines. It would have been highly interesting to analyse data for the remainder of 2020 to study the potential impacts of the COVID-19 pandemic. On a final note, it is worth mentioning that, with the exception of a few sporadic dips in the weekly rolling average-plot, the mean polarity scores remain positive across the entire period. This is perhaps in contrast to the general view of media always broadcasting bad news.

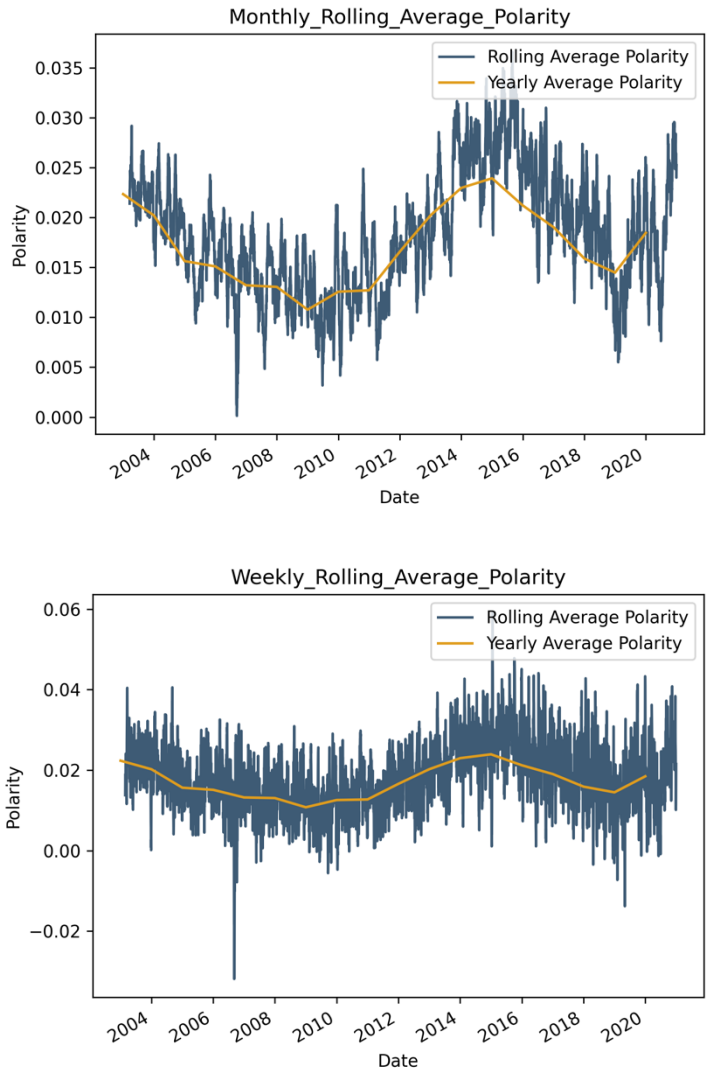

Figure 2 - Rolling averages of headline polarity scores

A4 – Network Analysis from Entity Co-occurrences

Link to the assignment folder (embedded in the overall repository) can be found here: https://github.com/bokajgd/Language-Analytics-Exam/tree/main/A4-Network-Analysis

Description

Network analysis refers to set of domain-agnostic methods for investigating the structure of relationships between object, places, or people. The approach is used across a plethora of fields, from mathematics and physics to social science and geospatial research. A network consists of nodes (representing the objects/people/places being analysed) that are connected by so called edges (representing the inter-node relationships) (Scott, 1988). A proper visualisation of a network graph often provides great insight into the relationships that are being examined, however, numerous metrics to quantify these relationships and structures can also beneficially be calculated.

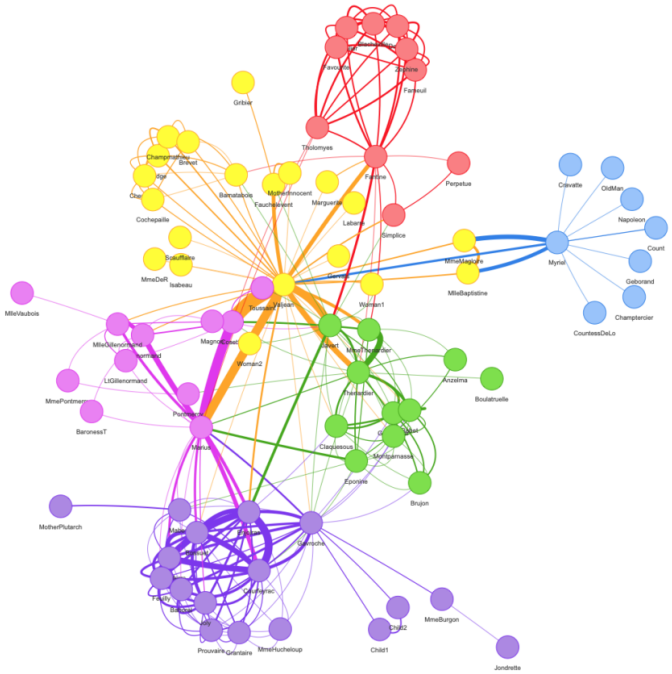

Figure 3 - Example of a network graph. Obtained from: https://www.r-bloggers.com/2019/06/interactive-network-visualization-with-r/

For this assignment, we were tasked with producing a stand-alone script capable of taking any weighted edgelist in a *.csv file format* (column headers 'nodeA', 'nodeB', 'weight') and creating and saving a network visualization based on this file. Furthermore, the script should generate and output a data frame showing the degree, betweenness, and eigenvector centrality scores for each node. As a bonus challenge, and since this course revolves around language analysis, I decided to add an extra feature enabling the user to perform a network analysis from scratch based on entity co-occurrences in a corpus of documents. More specifically, in addition to taking a pre-made edge list file, the script is able to take a *.csv file* containing a list of text documents, extract a user-defined category of entities from these texts using NER and then generate an edgelist based on entity co-occurrences in the

documents. As an example-case, I will use a data set of political news articles from the US to analyse the relationships between people mentioned in these articles based on article co-mentions. The data is obtained from this link.

Usage

If not already open, open a terminal window and redirect to the home folder of the cloned repository (see *General Instruction*). Remember to activate the virtual environment. Then, jump into the folder called *A2-Network-Analysis* using the following command:

```
cd A4-Network-Analysis
```

Now, it should be possible to run the following command in to get an understanding of how the script is executed and which arguments should be provided:

```
# Add -h to view which arguments should be passed
python3 src/A2-Network-Analysis.py -h

usage: A4-Network-Analysis.py [-h] [-tf --text_file] [-ef --edge_file]
                              [-id --identifier] [-ne --n_edges] [-tg --tag]

[INFO] Network Analysis of Entity Co-occurrences

optional arguments:
  -h, --help        show this help message and exit
  -tf --text_file   [DESCRIPTION] The name of the input text file
                    [TYPE]        str
                    [EXAMPLE]     -tf true_news.csv
  -ef --edge_file   [DESCRIPTION] The name of the input edge_file
                    [TYPE]        str
                    [EXAMPLE]     -fn edges_df.csv
  -id --identifier  [DESCRIPTION] Prefix for output files to identify files
                    [TYPE]        str
                    [EXAMPLE]     -id true_news
  -ne --n_edges     [DESCRIPTION] The number of edges to keep in the network.
                    [TYPE]        int
                    [DEFAULT]     50
                    [EXAMPLE]     -ne 50
  -tg --tag         [DESCRIPTION] Which entities to extract (LOC, PERSON, ORG)
                    [TYPE]        str
                    [DEFAULT]     PERSON
                    [EXAMPLE]     -tg PERSON
```

As can be seen from this help guide, the script can take several arguments. If no arguments are specified, the script loads in the example edgelist file, *edges_df.csv*, which is located in the *data/edge_files* folder. The number of edges to keep is set to 50 and the script is instructed by default to extract entities with the tag 'PERSON'. The output identifier label is set as the name of the given edgelist file (in this case *edges_df*).

Baseline execution of the script using the example edgelist file (generated in the notebook from class 6) can be performed using the following command line:

```
python3 src/A4-Network-Analysis.py
```

And this yields the same results as the following command:

```
python3 src/A4-Network-Analysis.py -fn edges_df.csv -id edges_df -ne 50 -tg PERSON
```

I highly recommend also testing the scripts full functionality by providing it with a raw text data set (*csv file* must contain a column with the header 'text'). An example data set is located in the *data/text_files* directory. The full dataset, *true_news.csv,* comprises more than 21.000 articles and, consequently, NER tagging takes a long time when passing this data. Thus, I recommend trying the script on *true_news_example.csv* which only contains a subset of 1.000 articles. The output discussed in the *Results* section is generated by running the following command:

```
python3 src/ A4-Network-Analysis.py -tf true_news_example.csv -ne 50 -tg PERSON
```

Feel free to play around with the *-ne* and the *-tg* parameters to create different plots (see *spaCy* (Honnibal & Montani, 2017) documentation for full list of tag categories).

Structure

The structure of the assignment folder can be viewed using the following command:

```
tree -L 2
```

This should yield the following graph:

```
.
├── README.md
├── data
│   ├── edge_files
│   └── text_files
├── output
│   ├── edges_df_centrality_df.csv
│   └── true_news_example_centrality_df.csv
├── src
│   └── A4-Network-Analysis.py
└── viz
    ├── edges_df_network_graph.png
    └── true_news_example_network_graph.png
```

The following table explains the directory structure in more detail:

Folder	Description
`data`	A folder containing the data that can be passed as input arguments to the main scripts: • *edge_files*: This subfolder contains all .csv edgelist files (also output directory for edge file created by script) • *text_files*: This subfolder contains raw text data sets in .csv format
`src`	A folder containing the source code (*A4-Network-Analysis.py*) created to solve the assignment.
`output`	An output folder in which the generated data frame containing centrality metrics for the nodes is saved • *edges_df_50_person_centrality_df.csv*: Centrality scores generated from *edges_df.csv* edgelist • *true_news_example_50_person_centrality_df.csv*: Centrality scores when running the script on the *true_news_example.csv* data with 'PERSON' set as tag • *true_news_example_30_gpe_centrality_df.csv*: Centrality scores when running the script on the *true_news_example.csv* data with 'GPE' set as tag and only the top 30 edges kept
`viz`	An output folder for the generated network visualisations • *edges_df_50_person_network_graph.png*: Network visualisation created from *edges_df.csv* edgelist • *true_news_example_50_person_network_graph.png*: Network visualisation created when running the script on the *true_news_example.csv* data with the 'PERSON' set as tag • *true_news_example_30_gpe_network_graph.png*: Network visualisation created when running the script on the *true_news_example.csv* data with 'GPE' set as tag and only the top 30 edges kept

Methods

Akin to the script in assignment 2, the script is coded using the principles of object-oriented programming. See the first paragraph of the *A2-Collocation* methods section for a quick outline of the general script architecture.

The script is highly dependent on the input file.
If the user passes a raw text data set, the script is set to perform the natural language processing steps needed for calculating document cooccurrence weights for a designated category of entities. Firstly, the desired entities are extracted for each document in input file using the *spaCy* library (Honnibal & Montani, 2017) for named entity recognition. By default,

the module extracts names of people and these people are what the nodes in the final network represent. However, this can be adjusted by the user (see *Usage*). For analyses of e.g., international diplomatic relationships via a corpus of United Nations documents, setting the tag to *GPE* would allow for the extraction of countries, cities, and states etc. If one had a corpus of football transfer rumour tweets, then setting the tag to *ORG* may enable insight into the trade connections between major European football clubs. When entities have been extracted for each document, all combinations of entity pairs that are mentioned together for each document is calculated and the frequencies of these pairs across the entire corpus are calculated. This co-mention frequency represents the weight of the *edge* that can be drawn between these two nodes (the two entities in the pair). The found edge weights for all node pairs are then combined into a single edgelist data frame using Pandas (McKinney et al., 2010) which can be used for visualising a network graph and calculating centrality scores. If, instead, a pre-made edgelist file is passed to the script, the previous steps are skipped and the script directly loads in the edge list. A network graph is then generated using the NetworkX (Aric et al., 2008) and Matplotlib (Hunter, 2007) libraries. The graph is drawn using a shell layout which scatters the nodes around a circle. In my opinion, this provides a neat and manageable network layout with minimal overlapping and from which is easy to locate nodes of high relevance. Two provide more details on the strength of the relationships in the network the widths of the edges are weighted according the 'weight' column in the edge list (number of document-cooccurrences). Likewise, the colours of the edges are also determined by this attribute. Adding both these visual features to the plot may be excessive, however, they neatly display the options one has when generating network visualisations. The user I also able to adjust the number of edges with the highest weights they whish to keep in the plot (too many insignificant edges spoil the plot). This is more generalisable and easier to manage than setting a minimum weight threshold in my opinion. Finally, the three requested centrality measures for each of the nodes are calculated and bundled into a data frame and saved to the output folder.

Results

The script fulfills the task objectives described in the *Description* and can be flexible used by the user from the command line. *Figure 4* shows the network graph generated when running the script with the following arguments:

```
python3 src/ A4-Network-Analysis.py -tf true_news_example.csv -ne 50 -tg PERSON
```

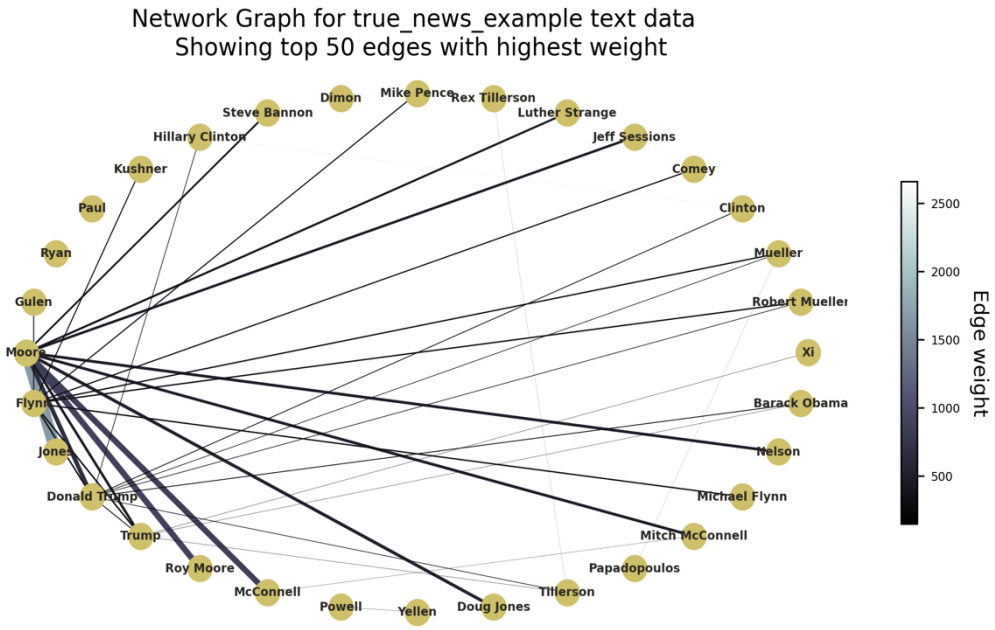

Figure 4 - Resulting network graph when running the script on the example text data provided in the data folder with 'PERSON' set as tag

Figure 5 shows the network graph generated when running the script with the following arguments:

```
python3 src/ A4-Network-Analysis.py -tf true_news_example.csv -ne 50 -tg PERSON
```

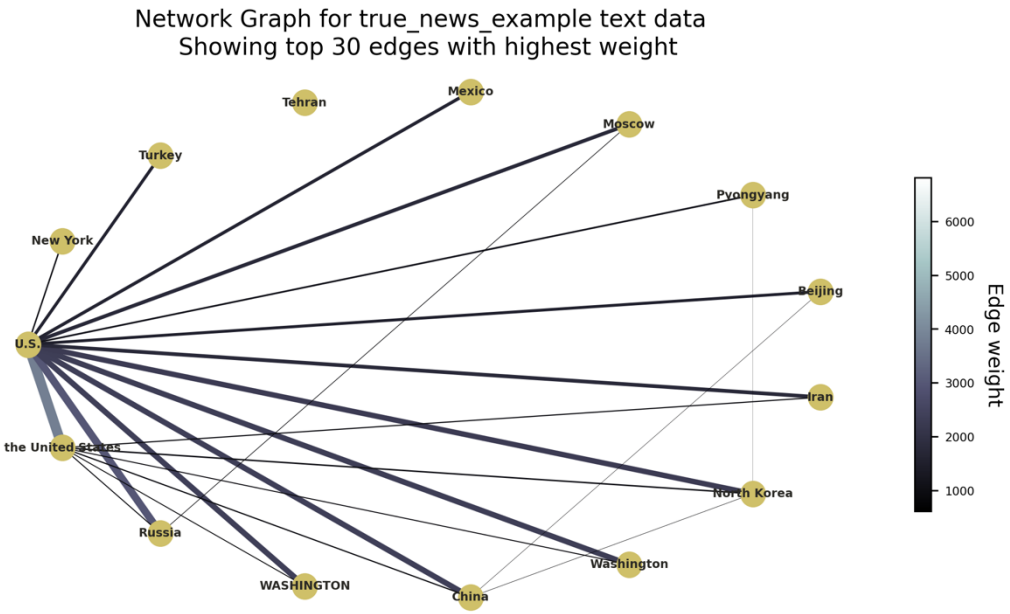

Figure 5 - Resulting network graph when running the script on the example text data provided in the data folder with 'GPE' set as tag

The title of the plot is adjusted according to the input arguments. Due to the possibility of errors and problems when installing and using the *Pygraphviz* package, I purposefully refrained from employing *NetworkX* graph layouts dependent on this package. This limited the number of possible plotting layouts.

Though the script works as intended, the network visualisation in *figure 2* reveals a minor problem. The first mention of a person in an article, often includes the full name. However, when this person is referred later in article, it is common practice to only mention the surname. Thus, the script interprets these two names as separate entities and creates two nodes even though they represent the same person (exemplified by strong connection between *Moore* and *Ryan Moore* nodes). Similarly, *figure 3* shows that many places can referred to using different names, e.g. *the United States* and *U.S.*. The script could, hence, be improved to take account for these node overlaps by using more complex NLP methods to identify and collapse entities that refer to the same concept/object/person.

A5 Supervised Learning - Readmission Prediction from Clinical Discharge Summaries

Disclaimer: A few code snippets in the *data_preprocessing.py* have been transferred and modified from scripts used for my bachelor's project on *Learning Latent Feature Representations of Clinical Notes for use in Predictive Models*. In general, however, the script is novel and coded specifically for this assignment.

Link to the assignment folder (embedded in the overall repository) can be found here: https://github.com/bokajgd/Language-Analytics-Exam/tree/main/A5-Supervised-Learning

Description

Digitalisation of health data documentation has caused a drastic increase in the amounts of electronic health records (EHRs) available for quantitative modelling. Combined with modern machine learning techniques, EHRs can be exploited to generate data driven tools for prediction and intervention in clinical health care settings (Jensen et al., 2012; Shickel et al., 2017). However, a majority of the stored data is in the form of unstructured, prosaic clinical notes. Extracting the valuable information embedded in these notes has proved highly challenging due their chaotic and sparse nature (Dubois et al., 2017).

In this study, I attempt to predict unplanned readmissions (within 30 days of discharge) of patients admitted to an intensive care unit using discharge notes written by nurses and physicians at the time of discharge. These notes contain a brief (~1000 words) summary of the hospitalisation and contain crucial qualitative information about patients that exceed what can be stored in structured data tables; expert insights, elaborations on procedures and symptoms and descriptions of social history. The notes are preprocessed and converted to feature representations using TF-IDF vectorisation. Subsequently, readmission labels are predicted using logistic regression. The simple, shallow nature of this algorithm allows me to calculate and visualize the direct influence of the input features.

One study estimates that expenses for unplanned readmissions yearly exceed $17 billion in the US alone (Jencks et al., 2009) while another concluded that, on average, 27% of unplanned readmissions are avoidable (van Walraven et al., 2011). Preventing such readmissions may help improve the quality of life for affected patients (Anderson & Steinberg, 1984) and prevent lives from being prematurely lost. It is clear that creating a tool which accurately foresees high readmission risk in patients may have positive implications for both patients, clinicians and spending. The 30-day time window is a common measure in unplanned readmission prediction (Kansagara et al., 2011).

Data

Electronic health records were obtained from the MIMIC-III database (Johnson et al., 2016) and the required data files were retrieved from PhysioNet (Goldberger et al., 2000). The MIMIC-III database is a freely accessible health-data resource which contains an extensive range of both structured and unstructured de-identified data from intensive care admissions

of more than 40.000 patients at the Beth Israel Deaconess Medical Center between 2001 and 2012. It comprises detailed information on demographics, test results, free-text clinical notes written by caregivers and ICD-9 procedure and diagnosis codes among others. The clinical notes dataset, which constitutes the foundation of the model exploration in this paper, comprises 2,083,180 human-written notes. The final cleaned discharge notes data used for training, validation and testing comrpise 43,876 notes.

Due to the high sensitivity of the data, the MIMIC-III database is, unfortunately, restricted and access to the data requires authorisation. Therefore, I am unable to share the data publicly in the folder and the code is, thus, not runnable on your machine. Please read through the code using the commenting as assistance for understanding the steps taken to complete the preprocessing and the supervised logistic regression classification model. Everyone is free to apply for access to the data from PhysioNet.

Usage

If one has access to the correct data in the data folder, executing the files could be carried out like this:

If not already open, open a terminal window and redirect to the home folder of the cloned repository (see *General Instruction*). Remember to activate the virtual environment. Then, jump into the folder called *A5-Supervised-Learning* using the following command:

```
cd A5-Supervised-Learning
```

Now, it should be possible to run the following command in to get an understanding of how the data pre-processing script is executed and which arguments should be provided:

```
python3 src/data_preprocessing.py -h
usage: data_preprocessing.py [-h] [-nf --notes_file] [-af --
admissions_file] [-mf --max_features] [-ng --ngram_range]

[INFO] Pre-processing discharge summaries

optional arguments:s
  -h, --help            show this help message and exit
  -nf --notes_file      [DESCRIPTION] The path for the file containing
clinical notes.
                        [TYPE]        str
                        [DEFAULT]     NOTEEVENTS.csv
                        [EXAMPLE]     -ne NOTEEVENTS.csv
  -af --admission_file  [DESCRIPTION] The path for the file containing
general admissions data
                        [TYPE]        str
                        [DEFAULT]     ADMISSIONS.csv
                        [EXAMPLE]     -ne ADMISSIONS.csv
  -mf --max_features    [DESCRIPTION] The number of features to keep in
the vectorised notes
                        [TYPE]        int
                        [DEFAULT]     30000
                        [EXAMPLE]     -mf 30000
  -ng --ngram_range     [DESCRIPTION] Defines the range of ngrams to
include (either 2 or 3)
                        [TYPE]        int
                        [DEFAULT]     3
                        [EXAMPLE]     -ng 3
```

By letting the script use the default inputs, the script can executed like this:

```
python3 src/ data_preprocessing.py
```

This outputs vectorised training, test and validation data along with classification labels (vector of 0 or 1s) and the vocabulary obtained when fitting the vectorised.

This data allows one to run the r*eadmission_prediction.py*. First, let us examine which arguments the script takes:

```
python3 src/readmission_prediction.py -h
usage: readmission_prediction.py [-h] [-tr --train_data] [-te --test_data]
                                 [-trl --train_labels] [--tel -test_labels]

[INFO] Readmission Prediction

optional arguments:
  -h, --help           show this help message and exit
  -tr --train_data     [DESCRIPTION] The file name of the training data csv
                       [TYPE]        str
                       [DEFAULT]     tfidf_train_notes.csv
                       [EXAMPLE]     -tr tfidf_train_notes.csv
  -te --test_data      [DESCRIPTION] The file name of the test data csv
                       [TYPE]        str
                       [DEFAULT]     tfidf_test_notes.csv
                       [EXAMPLE]     -tr tfidf_test_notes.csv
  -trl --train_labels  [DESCRIPTION] The file name of the training labels
csv
                       [TYPE]        str
                       [DEFAULT]     train_labels.csv
                       [EXAMPLE]     -tr train_labels.csv
  --tel --test_labels  [DESCRIPTION] The file name of the test labels csv
                       [TYPE]        str
                       [DEFAULT]     test_labels.csv
                       [EXAMPLE]     -tr test_labels.csv
```

Readmission prediction can now be performed using the following command:

```
python3 src/readmission_prediction.png
```

As the *NOTEEVENTS.csv* data file loaded in to the first script exceeds 4GB in storage, the script take around 15 minutes to execute on my MacBook.

Structure

The structure of the assignment folder can be viewed using the following command:

```
tree -L 2
```

This should yield the following graph:

```
.
├── README.md
├── data
│   ├── ADMISSIONS.csv
│   └── NOTEEVENTS.csv
├── output
│   ├── test_labels.csv
│   ├── tfidf_test_notes.csv
│   ├── tfidf_train_notes.csv
│   ├── tfidf_valid_notes.csv
│   ├── train_labels.csv
│   ├── valid_labels.csv
│   └── vocab
├── src
│   ├── data_preprocessing.py
│   └── readmission_prediction.py
└── viz
    ├── ROC-AUC.png
    ├── most_important.png
    └── preprocessing.png
```

The following table explains the directory structure in more detail:

Folder	Description
`data`	A folder containing the raw data that can be passed as input arguments to the preprocessing script: • ADMISSIONS.csv: This file contains all information and metadata on admissions • NOTEEVENTS.csv: This file contains all clinical notes written during all admissions
`src`	A folder containing the source code (*readmission_prediction.py* and *data_preprocessing.py*) created to solve the assignment.
`output`	An output folder in which the generated data frames containing vectorised notes, labels and the vocabulary used for matching influential features with their tokens • *.csv files*: Vectorised notes and labels for test, train and validation splits • *vocab*: This subfolder holds the vocabulary *vocabulary.pkl*
`viz`	An output folder for the generated visualisations and other visualisations for the README.md file • *ROC-AUC.png*: Image of AUC-ROC curve • *most_important.png*: Image showing the most influential features and the weigths • *preprocessing.png*: Flowchart of the preprocessing pipeline

Methods

Figure 6 displays the preprocessing steps from the raw data input to the final cleaned training, testing and validation data sets. This process from loading the raw data set to saving cleaned, split, vectorised data with appertaining labels is all performed in the *data_preprocessing.py* script. The script relies on the libraries *Numpy* (Harris et al., 2020) and *Pandas* (McKinney et al., 2010) for data handling and transformations, *nltk* (Bird et al., 2009) for string processing and *scikit-learn* (Pedregosa et al., 2011) for TF-IDF vectorization. Raw data on admissions (time, type etc.) and a *.csv* file containing the clinical notes are taken as input. All admissions with an associated discharge summary were retained and all notes except discharge summaries were discarded. For faulty instances with multiple discharge summaries, the note written first was kept. Admissions lacking a discharge summary (~3.9%) as well as 12 patients with overlapping admissions were removed. To preserve homogeneity in the dataset, 7863 neonatal admissions were also removed. Lastly, all admissions resulting in patient death were removed. The preprocessing is structured as follows:

1. Admission data is preprocessed and cleaned
2. Discharge summaries are extracted and preprocessed
3. Data frames are merged
4. Data is split into test (10%), validation (10%) and train (80%) sets
5. Training data is balanced by oversampling positive instances with a factor of 5 and then down-sampling negative instances to adjust. The general data is highly class-imbalanced and under 4% of cases lead to unplanned readmissions
6. Labels are extract and notes are TF-IDF vectorised and everything is saved to output folder

The training data set is used to train the final classification model. The validation dataset is used to explore which feature parameters during vectorization and hyperparameters in the logistic regression yield the highest possibel AUC-ROC score. The test set is kept completely hidden from the model until final performance evaluation to avoid bias. After manually testing classification performance on the validation set, the optimal number of max number of features to keep when vectorizing the notes was 30.000 (max_df = 0.8). A custom tokenizer function was designed to remove numbers and punctuation before tokenization. All English stopwords were further removed. Many medical terms are composed of multiple, conjoining words, e.g. *congestive heart failure*. Thus, to capture these word codependencies, notes were tokenized into both uni-, bi- and trigrams (also found to perform best with validation data).
The TF-IDF vectorised notes and classification labels output by the preprocessing script are taken as inputs to the *readmission_prediction.py* script. This script trains a *scikit-learn* based logistic regression classification model and tests its performance at predicting readmission on the test data set. The model uses *l2* regularization and a default *tolerance* of 0.0001. The script prints a classification report in the terminal window and generates a plot of the AUC-ROC plot which it saves in the viz folder. Lastly, the script calculates the top 20 features that drive the model most towards a negative prediction (no unplanned readmission) and positive prediction (unplanned readmission), respectively.

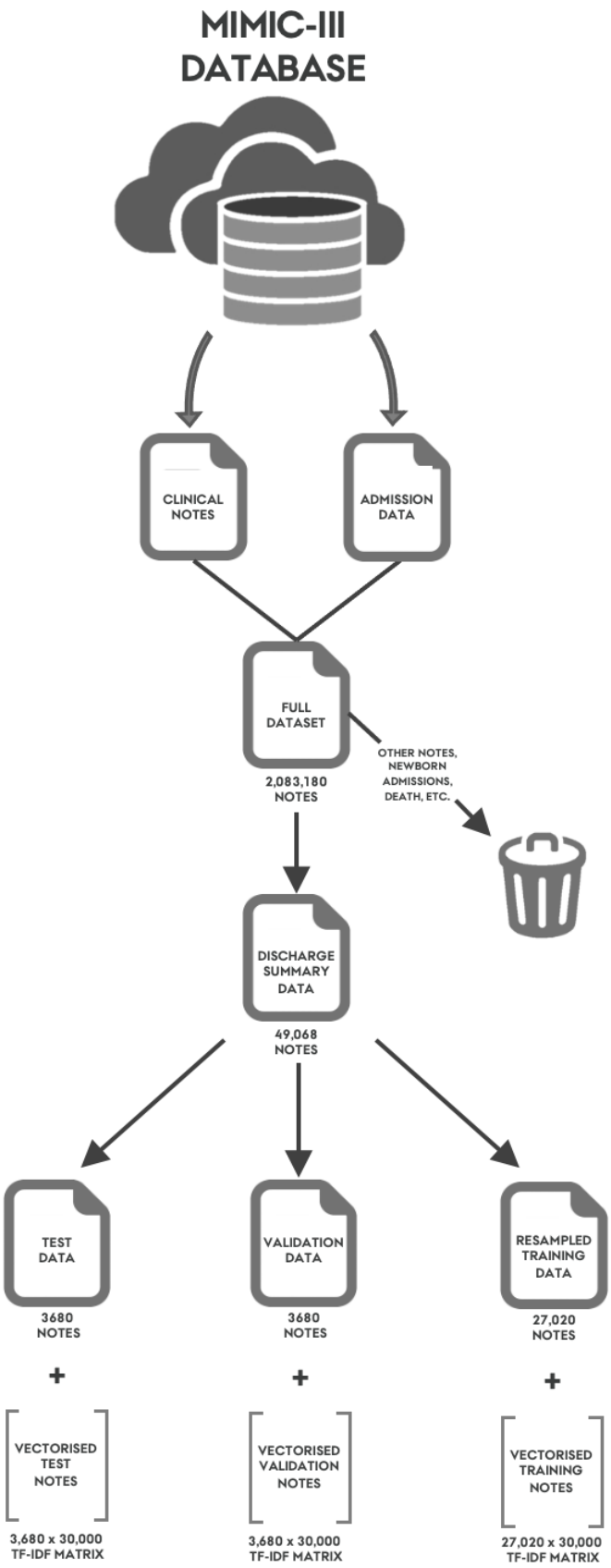

Figure 6 - A flowchart depicting the pre-processing pipeline

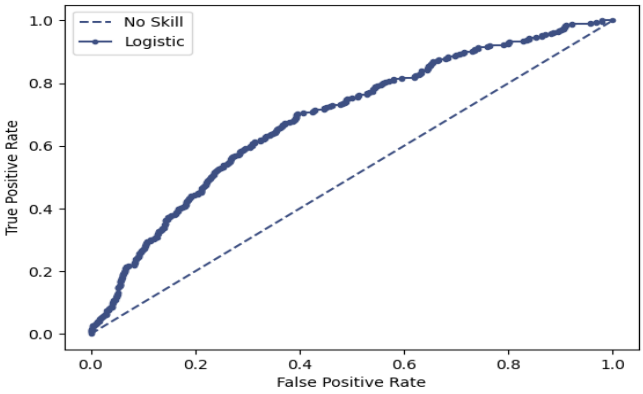

Figure 7 - AUC-ROC Curve

Discussion of Results

The logistic regression classifier trained on raw TF-IDF vectors yields an AUC-ROC of 0.698, a macro average recall of 0.62 and a macro average precision of 0.55 and overall accuracy of 0.78. As the test data (and real world scenarios) are highly imbalanced and contain very few positive instances, it is not meaningful to interpret the overall accuracy. AUC-ROC score is the main metric used for evaluation as this is the primary evaluation metric of many other readmission prediction publications (Rajkomar et al. 2018, Craig et al., 2017) and, therefore, it enables for convenient model comparisons. The score is designed to measure how well a model discriminates between positive and negative instances (Fawcett, 2006). As it is insensitive to class imbalance, it is regularly used when dealing with skewed data in binary prediction where basic accuracy evaluation would be worthless (Jeni et al., 2013). Despite its highly simple nature, the classifier performs similarly to other work aimed at predicting unplanned readmissions using discharge summaries; Craig et al. (2017) reports an AUC-ROC score of 0.71. *Figure* 7 shows the AUC-ROC curve for the classifier.

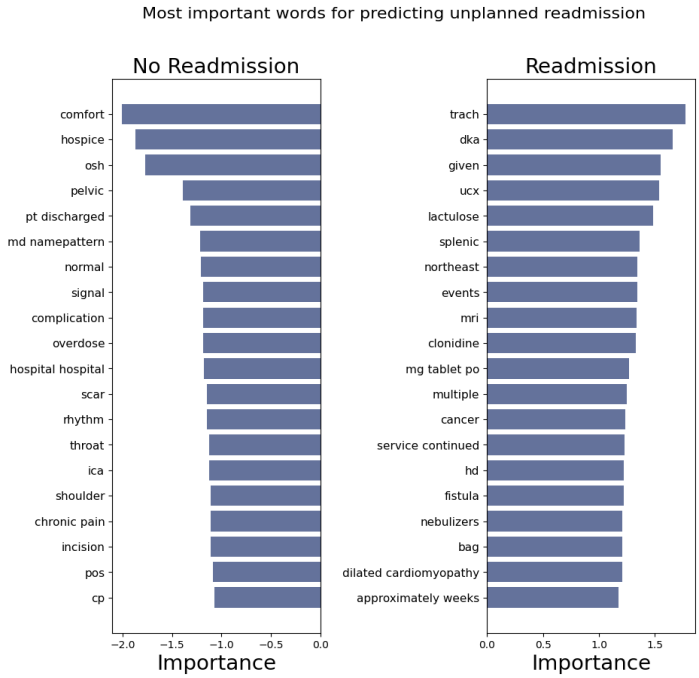

Figure 8 - Most Important Features

As stated in the description, the shallow architecture of logistic regression classification makes it easy to analyse of the influence of the individual input features. This helps guide model optimisation and increases transparency; understanding how the model derives its predictions is a crucial - especially in clinical settings. *Figure 8* shows the input tokens that push the model mostly towards predicting either 'no readmission' or 'unplanned readmission within 30 days'. This visualisation shows some interesting things. It is, for instance, sensible that *comfort* strongly drives the model towards believing that the patient is not exposed to being swiftly readmitted as they are feeling better. In the opposite end of the spectrum, the second word indicates that the patient is about to be transferred to a *hospice.* This is often a place people are transferred to when active care is no longer possible and death is near and, thus, patients are seldom readmitted hereafter. These results show that the model is capturing patterns/features in the notes that would, to a certain extent, also be influential to a human (perhaps a doctor) making their best guess.

In general, I believe that - if one wishes to maintain a simple model structure - feature parameters are the primary (and the most economical) knobs to turn for improving results. A point worth noting, is that using the AUC-ROC as the primary - and effectively - stand-alone evaluation metric is, in my view, highly problematic. The model clearly shows a low precision (expected with highly skewed data), but this is difficult to compare with other research as the measure is seldom reported. For a larger project, I would attempt to include the Matthews correlation coefficient (Chicco & Durman, 2020) for a study similar to this.

SA - Latent Feature Extraction Using LDA for Text Classification

Link to the assignment folder (embedded in the overall repository) can be found here: https://github.com/bokajgd/Language-Analytics-Exam/tree/main/SA-LDA-Feature-Extraction

Description

The vast majority of all human-generated data comes in the form of unstructured, free-written text; on the internet pages you visit daily, in the books you read at night or in the documents you process at work. However, when represented using classical Bag-of-Word feature representations, long prosaic texts are high-dimensional and extremely sparse and this makes it difficult to extract only the important information without being flooded by redundant stuff. Trying to model only meaningful features is denounced feature engineering and if often the key to good models. A common approach revolves around using NLP techniques to reduce sparsity and capture robust, latent representations of the input texts by transforming them into a lower-dimensional vector space.

For this self-assignment project, I wish to investigate using *Latent Dirichlet Allocation* (LDA) as a method for creating dimensionality-reduced latent feature representations that can be used as input to a classification model. LDA is a generative, unsupervsied topic modelling algorithm that constructs topics as bundles of words from a corpus and then represents each document as being composed of a distribution of these topics (Blei, 2003). This topic distribution vector can be extracted as a latent semantic representation of the document. I wish to investigate whether they are meaningfully usable as dense input features to a supervised text classification problem. See *figure 9* for a visual flowchart of how LDA works.

LDA as a means for generating dense input vectors for classification models is a rather unexplored method (Miotto et al., 2016; Phan et al., 2008) but I find it both intuitive and elegant and I believe that it is worth researching. In my opinion, this project neatly combines the principles of unsupervised learning and classical supervised classification that we have learned about during the semester.

Figure 9 - Simplified flowchart of how LDA transforms a document into a topic distribution vector

As a case study, I will employ my proposed method on a data set consisting of +45.000 fake and true news articles (balanced classes). This data was obtained from this link. An LDA model generating *n* topics (default 30) will be trained on a training set consisting of BoW vectorised articles in order to reduce all articles in the data set into an *n*-dimensional topic distribution vector. The training vectors will then train a logistic regression classifier aimed at predicting whether the article is fake or real and the performance hereof will be evaluated on an exclusive test set. A classifier trained on classic, sparse BoW feature vectors will also be trained and tested for comparison.

Usage

If not already open, open a terminal window and redirect to the home folder of the cloned repository (see *General Instruction*). Remember to activate the virtual environment. Then, jump into the folder called *SA-LDA-Feature-Extraction* using the following command:

```
cd SA-LDA-Feature-Extraction
```

Now, it should be possible to run the following command in to get an understanding of how the general script is executed and which arguments should be provided:

```
python3 src/lda_features_classification.py -h
usage: lda_features_classification.py [-h] [-pd --positive_data]
                                      [-nd --negative_data]
                                      [-mf --max_features] [-ng --ngram_range]
                                      [-ch --chunksize] [-pa --passes]

[INFO] Pre-processing discharge summaries

optional arguments:
  -h, --help           show this help message and exit
  -pd --positive_data  [DESCRIPTION] The path for the file containing data for
positive instances (true news stories).
                       [TYPE]        str
                       [DEFAULT]     True.csv
                       [EXAMPLE]     -pd True.csv
  -nd --negative_data  [DESCRIPTION] The path for the file containing data for
negative instances (false news stories)
                       [TYPE]        str
                       [DEFAULT]     False.csv
                       [EXAMPLE]     -nd FALSE.csv
  -mf --max_features   [DESCRIPTION] The number of features to keep in the
vectorised notes
                       [TYPE]        int
                       [DEFAULT]     30000
                       [EXAMPLE]     -mf 30000
  -ng --ngram_range    [DESCRIPTION] Defines the range of ngrams to include
(either 2 or 3)
                       [TYPE]        int
                       [DEFAULT]     3
                       [EXAMPLE]     -ng 3
  -ch --chunksize      [DESCRIPTION] The number of documents per chunk when
training the LDA model
                       [TYPE]        int
                       [DEFAULT]     200
                       [EXAMPLE]     -ch 200
  -pa --passes         [DESCRIPTION] The number of 'itereations' LDA training
should run for
                       [TYPE]        int
                       [DEFAULT]     10
                       [EXAMPLE]     -pa 10
  -nt --num_topic      [DESCRIPTION] The number of topics the LDA model should
generate and include in the topic feature vectors
                       [TYPE]        int
                       [DEFAULT]     30
                       [EXAMPLE]     -nt 30
  -wo --workers        [DESCRIPTION] The number of cores the LDA-MultiCore
model should use for training
                       [TYPE]        int
                       [DEFAULT]     3
                       [EXAMPLE]     -wo 3
```

This script provides a plethora of options for manually changing both the raw data input and as well adjusting parameters for the feature vectorisation and the LDA model training. Note that the user can adjust how many cores that LDA-MultiCore model should use when training using the *-wo* argument (default is 3). I recommend adjusting this to the highest

number allowed by your machine (-1) to speed up the execution. The most crucial parameter that one can flexibly adjust is the number of topics the LDA model should create (default 30). By letting the script use the default inputs, the script can be executed like this:

```
python3 src/lda_features_classification.py
```

Running the script prints two classification reports in the terminal and saves two confusion matrixes of the prediction results on the test set into the output folder.

Structure

The structure of the assignment folder can be viewed using the following command:

```
tree -L 2
```

This should yield the following graph:

```
.
├── README.md
├── data
│   ├── Fake.csv
│   ├── Fake_subset.csv
│   ├── True.csv
│   └── True_subset.csv
├── output
│   ├── BoW-Feature-Vectors_confusion_matrix.png
│   └── LDA-Topic-Vectors_confusion_matrix.png
├── src
│   └── lda_features_classification.py
└── viz
    └── lda.png
```

The following table explains the directory structure in more detail:

Folder	Description
`data`	A folder containing the raw data that can be passed as inputs to the Python script: • True.csv: This file contains all true news articles • Fake.csv: This file contains all fake news articles I have furthermore included two subsets of the data containing only 1000 articles each.

`src`	A folder containing the source code (*lda_features_classification.py*) created to solve the assignment.
`output`	An output folder in which the generated confusion matrices are saved: • *BoW-Feature-Vectors_confusion_matrix.png*: Confusion matrix for logistic regression trained directly on BoW feature vectors • *LDA-Topic-Vectors_confusion_matrix.png*: Confusion matrix for logistic regression trained on topic vectors generated using LDA
`viz`	An output folder for and other visualisations for the README.md file • *lda.png*: A flowchart of how LDA works

Methods

Similarly, to the other in assignments, the main script is coded using the principles of object-oriented programming. See the first paragraph of the *A2-Collocation* methods section for a quick outline of the general script architecture.

The main script can be subdivided into three separate steps:

1. Firstly, the input data is loaded, preprocessed (merged and split) and count-vectorised (also removing stopwords and punctuation etc.). As I love the simplicity and general structure of the vectorizer functions from *sci-kit learn* (Pedregosa et al,. 2011), I employ their *CountVectoriser()* for document vectorization and, subsequently, transform the feature vectors and vocabulary into *gensim*-manageable formats. 30.000 features are used for vectorization and both bi-grams and tri-grams are included. 20% of the data is reserved for unbiased model evaluation. Class balance for the training set is not strictly enforced but as the there is an equal amount of positive and negative cases in the entire corpus, the classes end up close to even.
2. Secondly, an LDA model is trained using *gensims* (Rehurek & Sojka, 2011) powerful *LdaMulticore()* function. After training, topic distribution vectors are obtained for both the training articles and the test articles.
3. Lastly, two logistic regression classification models are trained and tested using *scikit-learn* functions. One is trained and tested using the sparse 30k-dimensional BoW vectors, the other using the LDA-generated topic distribution vectors. A confusion matrix showing the performance of each these models is saved to the *output* folder and a classification report is .

For ideal research practices, it would be sensible to reserve a subset of the data as a validation which can be used for finetuning hyperparameters etc. However, as this is simply a toy case study for exploring the technique, simply evaluating the models once on a test set that it has not previously been exposed to will suffice.

Discussion of Results

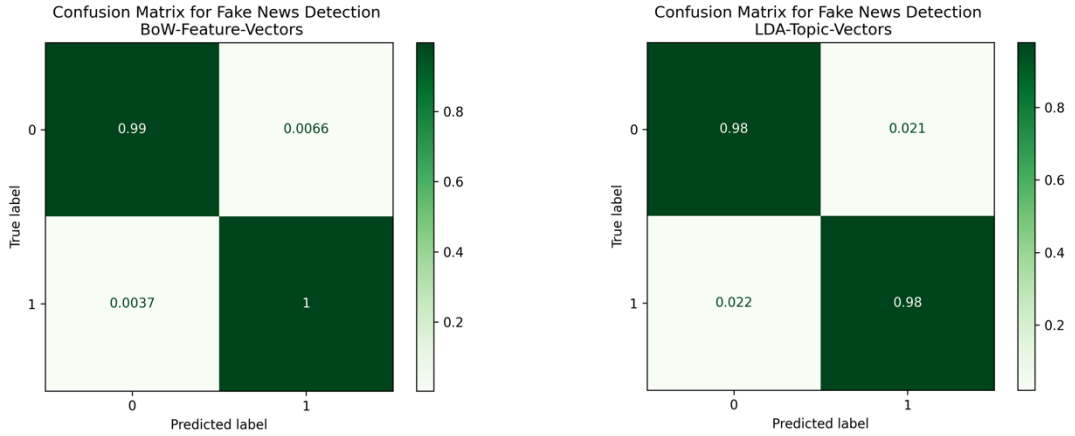

Figure 9 displays the resulting confusion matrices output by the script when using the default parameters settings (30 LDA topics). From immediate glance, it is evident that this is an easy classification task and both models yield close to perfect results. The model trained on sparse BoW vectors slightly outperforms the LDA based features and with such high accuracies the data set may be a poor case study for analyzing the success of the technique.

However, I must restate that the original incentive of this project was not purely to attempt to outperform a BoW model. It is expected that a dimensionality reduction with a factor 1000 would lead to a loss in meaningful information when compared to a model that performs with an accuracy of >99% on a simple logistic regression classifier using sparse BoW vectors. The incentive was more accurately, to investigate, how much discriminative power the reduced topic vectors were able to preserve. Therefore, if you turn it around, the results are actually quite remarkable; we have reduced the input variable from 30.000 meaningful features (sparse, of course) into a feature space with only 30 dimensions and still upheld an accuracy of 98%.

One of the key forces of using topic modelling for dimensionality reduction and feature engineering is that it is easy to combine with other feature engineering techniques for other data types. In classification problems, where a number of different data types need to be appended to a single input feature vector, it is convenient to reduce the textual data into a dense and compact representation. A sparse BoW representation of textual data may be powerful in a stand-alone model but could drown other more compact features when combined with more data types. Miotto et al. (2016) utilize this trait in their attempts in their attempt to model future diseases. They compress all clinical notes for a patient into a 300-dimensional aggregated topic distribution vector using LDA. This dense feature vector is then concatenated with other features such as medication dosages and diagnosis codes.

Critical Evaluation

Obviously, this one case study on a simple fake news-detection task that yields close to perfect performances is completely insufficient to make any robust inferences about the value of using LDA for generating latent, dense representations as input vectors to classification models. The method should be tested thoroughly across a number of use

cases and in scenarios with varying number of outcome classes and different basis for performance outcomes. It would be interesting to test the method on documents of longer length, e.g. novels or law texts or in predication cases where multiple data types can beneficially be concatenated.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

Anderson, G. F., & Steinberg, E. P. (2010, January 13). Hospital Readmissions in the Medicare Population (world)

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

Baker, P. (2006). Using corpora in discourse analysis. A&C Black.

Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. the Journal of machine Learning research, 3, 993-1022.

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC genomics, 21(1), 1-13.

Craig, E., Arias, C., & Gillman, D. (2017). Predicting readmission risk from doctors’ notes. ArXiv:1711.10663 [Stat]

Dubois, S., Romano, N., Kale, D. C., Shah, N., & Jung, K. (2017). Learning effective representations from clinical notes. stat, 1050, 15.

Evert, S. (2010). Computational Approaches to Collocations. www.collocations.de

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Feldman, R. (2013). Techniques and applications for sentiment analysis. Communications of the ACM, 56(4), 82-89.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. Studies in linguistic analysis.

Goldberger Ary L., Amaral Luis A. N., Glass Leon, Hausdorff Jeffrey M., Ivanov Plamen Ch., Mark Roger G., Mietus Joseph E., Moody George B., Peng Chung-Kang, & Stanley H. Eugene. (2000). PhysioBank, PhysioToolkit, and PhysioNet. Circulation, 101(23), e215–e220.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. IEEE Annals of the History of Computing, 9(03), 90-95.

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

Jencks, S. F., Williams, M. V., & Coleman, E. A. (2009). Rehospitalizations among Patients in the Medicare Fee-for-Service Program. New England Journal of Medicine, 360(14), 1418–1428.

Jeni, L. A., Cohn, J. F., & De La Torre, F. (2013). Facing imbalanced data—Recommendations for the use of performance metrics. 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, 245–251.

Jensen, P. B., Jensen, L. J., & Brunak, S. (2012). Mining electronic health records: Towards better research applications and clinical care. Nature Reviews Genetics, 13(6), 395–405.

Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., & Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. Scientific Data, 3.

Loria, S. (2018). textblob Documentation. Release 0.15, 2.

Miotto, R., Li, L., Kidd, B. A., & Dudley, J. T. (2016). Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. Scientific reports, 6(1), 1-10.

McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.

Phan, X. H., Nguyen, L. M., & Horiguchi, S. (2008, April). Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In Proceedings of the 17th international conference on World Wide Web (pp. 91-100).

Rajkomar, A., Oren, E., Chen, K., Dai, A. M., Hajaj, N., Hardt, M., Liu, P. J., Liu, X., Marcus, J., Sun, M., Sundberg, P., Yee, H., Zhang, K., Zhang, Y., Flores, G., Duggan, G. E., Irvine, J., Le, Q., Litsch, K., ... Dean, J. (2018). Scalable and accurate deep learning with electronic health records. Npj Digital Medicine, 1(1), 1–10.

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2).

Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2017). Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE journal of biomedical and health informatics, 22(5), 1589-1604.

Scott, J. (1988). Social network analysis. Sociology, 22(1), 109-127.

Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace.

Van Walraven, C., Bennett, C., Jennings, A., Austin, P. C., & Forster, A. J. (2011). Proportion of hospital readmissions deemed avoidable: a systematic review. Cmaj, 183(7), E391-E402.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2019). HuggingFace's Transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.