Language Analytics portfolio

Emil Trenckner Jessen

27th May 2021

Contents

1 Introduction **3**
1.1 GitHub repository . . . 3
1.2 Getting started . . . 3

2 Sentiment analysis (assignment 3) **3**
2.1 Assignment description . . . 3
2.2 Methods . . . 4
2.3 Results and discussion . . . 5
2.4 Usage . . . 7
2.4.1 Optional arguments . . . 7

3 Network analysis (assignment 4) **7**
3.1 Assignment description . . . 7
3.2 Methods . . . 7
3.3 Results and discussion . . . 8
3.4 Usage . . . 10
3.4.1 Optional arguments . . . 10

4 LDA and topic modeling on philosophical texts (assignment 5) **10**
4.1 Assignment description . . . 10
4.2 Methods . . . 11
4.3 Results and discussion . . . 12
4.4 Usage . . . 14
4.4.1 Optional arguments . . . 14

5 Text classification using Deep Learning (assignment 6) **14**
5.1 Assignment description . . . 14
5.2 Methods . . . 15
5.3 Results and discussion . . . 15
5.4 Usage . . . 17
5.4.1 Optional arguments . . . 17

6 LSTM models for text generation (assignment 7 - self-assigned) **17**
6.1 Assignment description . . . 17
6.2 Methods . . . 18
6.3 Results and discussion . . . 19
6.4 Usage . . . 22
6.4.1 Optional arguments . . . 22

7 Acknowledgements **22**

1 Introduction

This document provides an overview of the portfolio in the Cultural Data Science course Language Analytics. The document is divided up with sections for the individual assignments that contains subsections providing information on 1) The assignment description, 2) Methods and 3) Results and discussion.

1.1 GitHub repository

Access to the GitHub repository can be found via the link here:

https://github.com/emiltj/cds-language-exam

The repository contains the individual assignments a long with READMEs. The READMEs specify cloning the repository, setup (virtual environment installation and data collection) as well as the execution of the individual assignments. The information is also provided in this document.

1.2 Getting started

For running my scripts I'd recommend following the below steps in your bash-terminal (notice the bash scripts are different, depending on your OS). This functions as a setup of the virtual environment, as well as an execution of a bash script that downloads all the data to the data folders respective to the assignments.

Cloning repository and creating virtual environment

Listing 1: bash terminal - MAC/LINUX/WORKER02

```
git clone https://github.com/emiltj/cds-language-exam.git
cd cds-language-exam
bash ./create_lang_venv.sh
```

Listing 2: bash terminal - WINDOWS

```
git clone https://github.com/emiltj/cds-language-exam.git
cd cds-language-exam
bash ./create_lang_venv_win.sh
```

Retrieving the data

The data is not contained within this repository, considering the sheer size of the data. Using the provided bash script data_download.sh that I have created, the data will be downloaded from a Google Drive folder and automatically placed within the respective assignment directories.

Listing 3: bash terminal

```
bash data_download.sh
```

After cloning the repository, creating the virtual environment and retrieving the data you should be ready to go. Move to the assignment directories and read the READMEs for further instructions.

2 Sentiment analysis (assignment 3)

2.1 Assignment description

Use the data set "A million headlines" - headlines from the Australian news source ABC (Start Date: 2003-02-19 ; End Date: 2020-12-31).

Do the following:

- Calculate the sentiment score for every headline in the data. You can do this using the spaCyTextBlob approach that we covered in class or any other dictionary-based approach in Python.
- Create and save a plot of sentiment over time with a 1-week rolling average
- Create and save a plot of sentiment over time with a 1-month rolling average
- Make sure that you have clear values on the x-axis and that you include the following: a plot title; labels for the x and y axes; and a legend for the plot
- Write a short summary (no more than a paragraph) describing what the two plots show. You should mention the following points: 1) What (if any) are the general trends? 2) What (if any) inferences might you draw from them?

2.2 Methods

Specifically for this assignment

The script first converts the dates to datetime format and then calculates the sentiment scores for each of the headlines. To calculate sentiment scores, I use the SpaCy model en_core_web_sm. Sentiment analysis using this approach not only rates sentences as being positive or negative, but rather applies scores based on the structure and wording of the sentence. It takes into account word context so that intensifiers may strengthen or weaken the sentiment of a given word - i.e. giving stronger negative score for "I am very angry" as opposed to "I am angry". In this example, the word "very" acts as an intensifier, despite being a neutrally charged word by itself (read more here). After calculating the sentiment scores they are then averaged for each day. The signal of averaged sentiment scores is quite noisy and it can be hard to discern any long term low frequency patterns. By smoothing or applying a low-pass filter, we filter away the high frequency oscillations of the signal. This allows for better seeing the more general, long ranging patterns of the data. With this as rationale, the daily sentiment scores are smoothed, by applying rolling windows of window sizes 7 and 30 (weekly and monthly smoothing), with a step size of 1. For creating the plots, I used matplotlib and included plots on the scores for each day both smoothed and unsmoothed. I also chose to have a plot of all 4 subplots in a single plot to provide a clear overview of the increase in interpretability that smoothing enable.

On a more general level (this applies to all assignments)

I have tried to as accessible and user-friendly as possible. This has been attempted by the use of:

- Smaller functions. These are intended to solve the sub-tasks of the assignment. This is meant to improve readability of the script, as well as simplifying the use of the script.
- Information prints. Information is printed to the terminal to allow the user to know what is being processed in the background.
- Argparsing. Arguments that let the user determine the behaviour and paths of the script.

2.3 Results and discussion

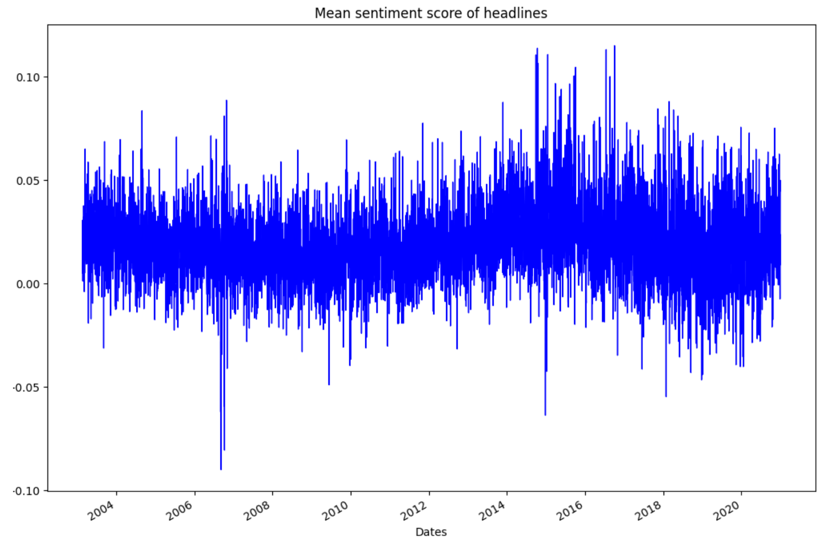

Figure 1: Sentiment scores over time, no smoothing

When looking at the non-smoothed sentiment scores of the ABC news articles it can be hard to find any general patterns due to the great fluctuation that is apparent on a daily basis. To be able to discern any patterns, we need to extract information from the noisy signal by attenuating the higher frequency components of this signal.

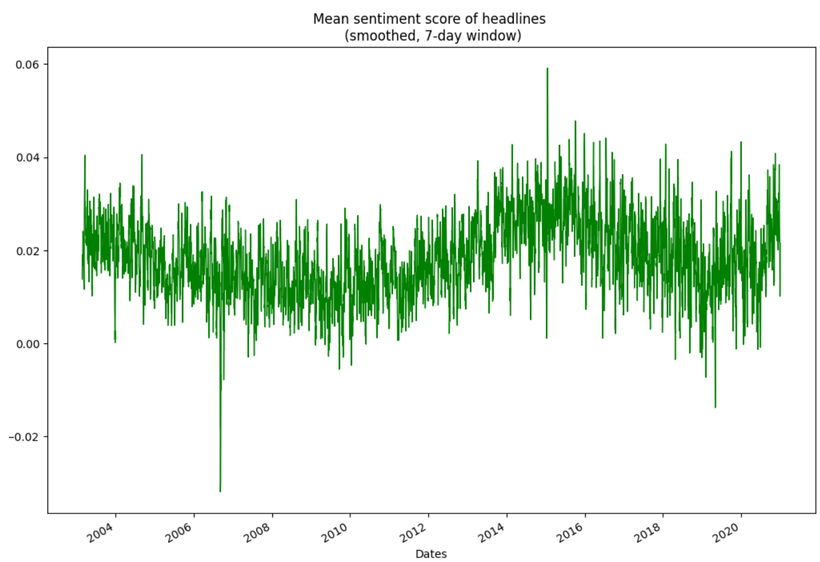

Figure 2: Sentiment scores over time, 7-day smoothing

Looking at the signal once smoothed with the 7-days moving averaging window, some trends seem to be discernable; spike in positivity around 2015-16 and a drop around 2019.

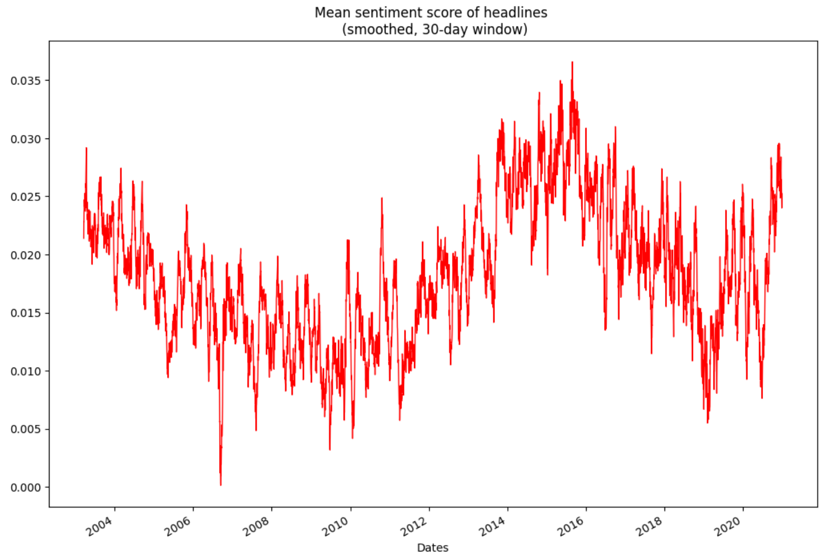

Figure 3: Sentiment scores over time, 30-day smoothing

Looking at the smoothed signal once smoothed over a 30-day period, the trend described above seem to be even easier to see. Do note that the values on the y-axis also change, due to outliers having less of a say when averaging across multiple days.

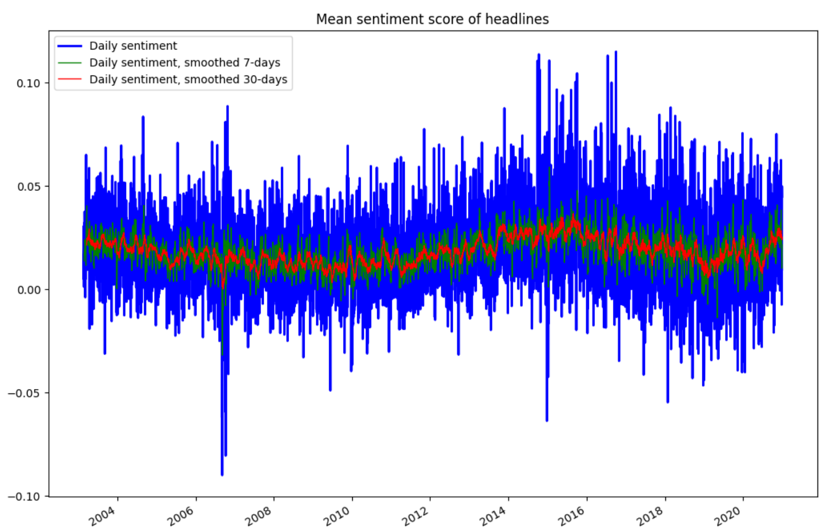

Figure 4: Sentiment scores over time, combined

In general, when looking at the smoothed signal, we can start to detect low-frequency patterns in the oscillations as the higher frequencies are attenuated. It can be hard to make any inferences as to what might have caused the spike in positivity around the year 2015-16 and similarly also hard to explain the drops around the year 2010 as many factors play in. However, given the horrible bush fires that killed hundreds of people and occurred in late 2019 and early 2020 it does not come as a surprise that there is a large drop in positive sentiment in the news articles in the Australian news around that time.

2.4 Usage

Make sure to follow the instructions in the README.md located at the parent level of the repository, for the required installation of the virtual environment as well as the data download.

Subsequently, use the following code (when within the cds-language-exam folder):

Listing 4: bash terminal

```
cd assignment_3
source ../lang101/bin/activate # If not already activated
python sentiment.py
```

2.4.1 Optional arguments

- '-i', '–inputpath', type = str, default = os.path.join("data", "abcnews-date-text.csv"), required = False, help = f"str - path to .csv. n")
- '-t', '–test', type = bool, default = False, required = False, help = 'bool - if True, then performs only on a subset. False is on the full dataset')

3 Network analysis (assignment 4)

3.1 Assignment description

This command-line tool will take a given data set and perform simple network analysis. In particular, it will build networks based on entities appearing together in the same documents, like we did in class.

Your script should be able to be run from the command line It should take any weighted edgelist as an input, providing that edgelist is saved as a CSV with the column headers "nodeA", "nodeB" For any given weighted edgelist given as an input, your script should be used to create a network visualization, which will be saved in a folder called viz. It should also create a data frame showing the degree, betweenness, and eigenvector centrality for each node. It should save this as a CSV in a folder called output.

- Your script should be able to be run from the command line
- It should take any weighted edgelist as an input, providing that edgelist is saved as a CSV with the column headers "nodeA", "nodeB"
- For any given weighted edgelist given as an input, your script should be used to create a network visualization, which will be saved in a folder called viz.
- It should also create a data frame showing the degree, betweenness, and eigenvector centrality for each node. It should save this as a CSV in a folder called output.

3.2 Methods

Specifically for this assignment

A prerequisite for completing this assignment is having a weighted edgelist. I have therefore decided to include an additional script, which generates a weighted edgelist (*create_edgelist.py*). This script takes the *fake_or_real_news.csv* dataset and extracts its entities with the label [PERSON]. It utilizes the model en_core_web_sm from the SpaCy library. It then find entity pairs (entities that appear within the same document) and counts how often these pairs have appeared in all news articles - these counts are the weight of each of the unique pairs. The weighted edgelist is then saved as a .csv.

The actual assignment script *network.py* takes the newly created weighted edgelist as input. The argument *-n* specifies how node pairs the network analysis should include. If n is less than the number total node pairs, it will filter away the lowest weighted pairs. It plots the network using the package *networkx* and saves it to directory *viz*. It also calculates centrality measures and saves it as a .csv in the folder *output*. The measures are eigenvector centrality, betweenness centrality and degree centrality. Eigenvector centrality is a measure of influence of a node - nodes with many connections to other well connected nodes will have higher scores. Betweenness centrality is a measure of centrality in a network - a node that lies on communication flows can control the flow. Calculated by computing the shortest paths between all nodes, then determining the fraction of the number of these paths that go through a given node in question, compared to total number of paths. In a weighted network such as this one, scores are higher given higher edge weights. Degree centrality is merely the number of connections a given node has.

On a more general level (this applies to all assignments)

I have tried to as accessible and user-friendly as possible. This has been attempted by the use of:

- Smaller functions. These are intended to solve the sub-tasks of the assignment. This is meant to improve readability of the script, as well as simplifying the use of the script.
- Information prints. Information is printed to the terminal to allow the user to know what is being processed in the background.
- Argparsing. Arguments that let the user determine the behaviour and paths of the script.

3.3 Results and discussion

Creating an edgelist

	nodeA	nodeB	weight
0	John F. Kerry	Kerry	21
1	John F. Kerry	Laurent Fabius	2
2	Francois Hollande	John F. Kerry	1
3	John F. Kerry	Obama	76
4	Benjamin Netanyahu	John F. Kerry	7
5	Jane Hartley	John F. Kerry	1

Table 1: Excerpt from the generated edgelist

As can be seen in the table above, the script for generating weighted edgelists has been successful in that it indeed has created a weighted edgelist. The entity extraction of people has correctly both identified John F. Kerry and Kerry as people entities. As can be seen in the table however, the script was not programmed to merge entities referring to the save person into a single entity. Kerry and John F. Kerry likely refer to the same person, but appear as to seperate entities in the edgelist. Additional processing ought to have been carried out to circumvent this problem.

Network analysis

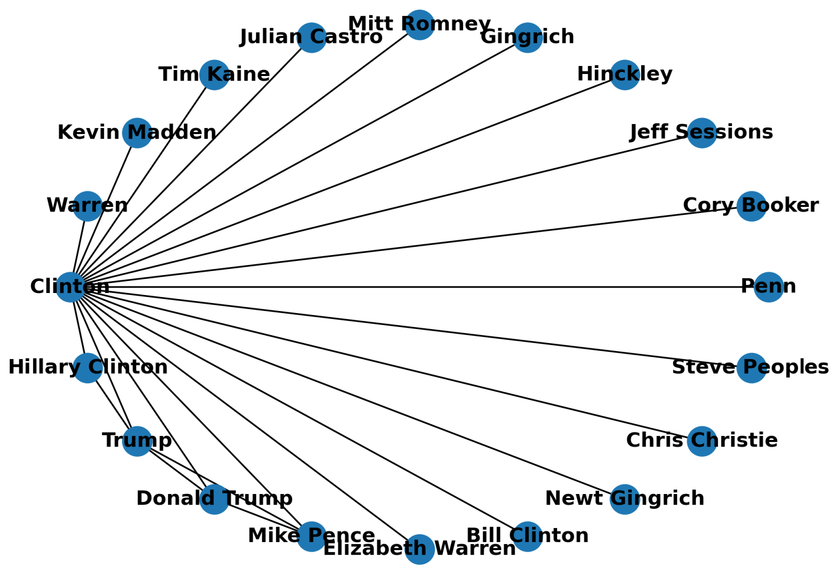

Figure 5: The network visualized - showing the 15 highest weighted connections

When looking at the visualization of the network of the 15 strongest connections (pairs to include is n = 15 as default, when not otherwise specified using the optional arguments), it appears that Hillary Clinton, Barack Obama and Donald Trump have some of the strongest connections. However, it should of course be noted that a large portion of strong connections between to entities that refer to the same person. This is a result of the edgelist used as input. Otherwise, the script seems to produce the desired outcome.

	node	eigenvector_centrality	betweenness_centrality	degree_centrality
0	Clinton	0.5276992157079976	0.7087912087912088	0.8571428571428571
1	Trump	0.5359784095931499	0.4670329670329671	0.6428571428571428
2	Obama	0.2877321039031794	0.0	0.2857142857142857
3	Hillary Clinton	0.22647028527916035	0.0	0.14285714285714285
4	Bush	0.23017534121767486	0.14285714285714288	0.3571428571428571

Table 2: Excerpt from the centrality measures output

As can be seen in the table above, the problem identified in the creation of the edgelist leaks through - when looking at the excerpt Clinton appears twice. Does Clinton refer to Bill Clinton or Hillary? Or perhaps sometimes Bill and other times Hillary? We cannot know for sure. Regardless when looking at eigenvector centrality, it seems that Trump and Clinton have many connections to other highly connected people. When looking at betweenness centrality, it appears that Clinton functions as a link between a lot of other nodes, glueing many people together. When looking at degree centrality - sheer number of connections, Clinton and Trump appears at the clear top.

Important to note though; the table shown above is merely a small excerpt of the full table. The excerpt has been chosen as to clarify the principles of analysis, rather than to do a full-scale analysis It is also worthwhile mentioning that the exact scores are not to be trusted too much, due to the same people appearing as multiple nodes (e.g. Clinton, Hillary Clinton, etc.). The script producing this table, seems to work according to the desired outcome. Another edgelist should have been used to produce a proper in-depth analysis.

3.4 Usage

Make sure to follow the instructions in the README.md located at the parent level of the repository, for the required installation of the virtual environment as well as the data download.

Subsequently, use the following code (when within the cds-language-exam folder):

Listing 5: bash terminal

```
cd assignment_4
source ../lang101/bin/activate # If not already activated
python create_edgelist.py
python network.py
```

3.4.1 Optional arguments

create_edgelist.py arguments for commandline to consider:

- '-i', '–inpath', type = str, default = os.path.join("data", "fake_or_real_news.csv"), Default when not specifying a path required = False, help = "Inputpath for generating edgelist")

network.py arguments for commandline to consider:

- "-i", "–inpath", type = str, default = os.path.join("out","weighted_edgelist.csv"), Default when not specifying a path required = False, Since we have a default value, it is not required to specify this argument help = "str containing path to edgelist file")
- "-n", "–n", type = int, default = 25, Default when not specifying anything in the terminal required = False, Since we have a default value, it is not required to specify this argument help = "int specifying number of node + edge pairs wanted in the analysis (top n weighted pairs)")

4 LDA and topic modeling on philosophical texts (assignment 5)

4.1 Assignment description

General assignment description

Pick your own data set to study. Train an LDA model on your data to extract structured information that can provide insight into your data. For example, maybe you are interested in seeing how different authors cluster together or how concepts change over time in this data set. You should formulate a short research statement explaining why you have chosen this data set and what you hope to investigate. This only needs to be a paragraph or two long and should be included as a README file along with the code. E.g.: I chose this data set because I am interested in... I wanted to see if it was possible to predict X for this corpus. You should also include a couple of paragraphs in the README on the results, so that a reader can make sense of it all. E.g.: I wanted to study if it was possible to predict X. The most successful model I trained had a weighted accuracy of 0.6, implying that it is not possible to predict X from the text content alone. And so on.

Tips

- *Think carefully about the kind of preprocessing steps your text data may require - and document these decisions!*

- *Your choice of data will (or should) dictate the task you choose - that is to say, some data are clearly more suited to supervised than unsupervised learning and vice versa. * Make sure you use an appropriate method for the data and for the question you want to answer*
- *Your peer reviewer needs to see how you came to your results - they don't strictly speaking need lots of fancy command line arguments set up using argparse(). You should still try to have well-structured code, of course, but you can focus less on having a fully-featured command line tool*

Instructions for my specific project

This assignment seeks to use Latent Dirichlet Analysis (LDA) as a tool for performing topic modeling. It investigates historical philosophical texts from different schools of philosophical thought in an exploratory manner. More specifically, it seeks to investigate whether particular schools of philosophical have similarities in terms of topics. Thereby seeking answers to questions such as: *Do texts from German Idealism incorporate the same topics as Nietzsches texts?* and *Do old Greek philosophical schools cluster together in terms of topics?*

Moreover, instead of just looking at schools as one homogeneous group of titles, I also want to look at the individual books within each school. *Are books that follow the same philosophical school similar in terms of topics?*

- Merge paragraphs from the same books together in the philosophical text corpus.
- Perform LDA, using bigram and trigram models. Ensure that the LDA utilizes K=5 topics
- Create a visualization that depicts each philosophical schools' respective topic prevalence.
- Reduce the 5-dimensional topic space to 2 dimensions using Principal Component Analysis (PCA)
- Plot the schools in this PCA-space (with X and Y axes showing principal component 1 and 2, respectively)
- Plot the individual books in this PCA-space (with X and Y axes showing principal component 1 and 2, respectively)

4.2 Methods

Specifically for this assignment

For this assignment, I first aggregated all entries from the same book title together, to have the entire text of one book as one entry. I did this as I wanted to look at individual books for my visualizations. I then built bigram and trigram models which would find contiguous sequences of 2 or 3 items (phrases of 2 or 3 words). The models had a threshold score of 100, which meant that they would only allow phrases if the score of the phrase was greater than the threshold. Using the models, I processed the entries (i.e. books), only keeping nouns, verbs, adverbs, and adjectives. I then created a dictionary so I could convert the processed data into vectors. A word in an entry would then be converted into an integer value (with the value functioning like an ID for the dictionary). Using the processed corpus I built an LDA model with a default of K=5 topics. As the number of topics is mostly an arbitrary choice, I let the user be enabled to specify another number, using the argument *–ntopics*. A perplexity score (a statistical measure of how well the model predicts a sample) and a coherence score (measure of the degree of semantic similarity between high scoring words in the topic) are printed to the terminal upon running the script. The prevalences of the topics are then computed for each book title. Using PCA, the 5 dimensions (one for each topic) are reduced to 2 dimensions. All books are then plotted in this PCA-space and colored by the respective school that they belong to. The topic prevalences of all books from the same philosophical school are then averaged together, to get a simplistic measure of topic prevalence in each philosophical school. A plot is then created that show the topic prevalence distribution across schools of philosophical thought. To be able to see if some schools have a similar topic distribution more clearly, PCA is once again utilized. The average topic prevalences for each school are plotted in this PCA-space.

Finally, I also saved an interactive .html document that shows the intertopic distance also using PCA. This also gives lists of the most important words for each topic.

On a more general level (this applies to all assignments)

I have tried to as accessible and user-friendly as possible. This has been attempted by the use of:

- Smaller functions. These are intended to solve the sub-tasks of the assignment. This is meant to improve readability of the script, as well as simplifying the use of the script.
- Information prints. Information is printed to the terminal to allow the user to know what is being processed in the background.
- Argparsing. Arguments that let the user determine the behaviour and paths of the script.

4.3 Results and discussion

Given the exploratory nature of this assignment few quantitative results have been generated. As a consequence, this section will rather provide an overview and interpretation of the visual output.

Please do note that this section looks at the results for K = 5. As performance metrics can be compared across different K's (numbers of topics) using the argument *–ntopics*, one may want to experiment finding the optimal number of topics.

Topic prevalence in schools of philosophical thought

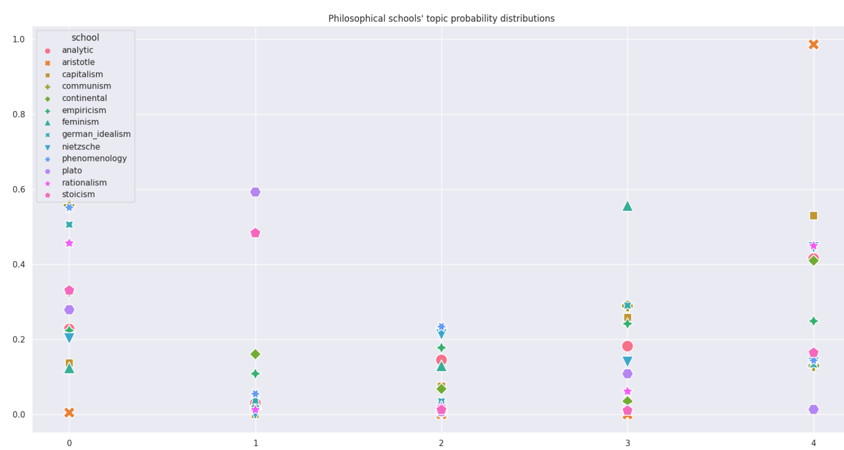

Figure 6: The topic prevalence of the different schools of philosophical thought

When looking at the schools' topic prevalences the first thing that comes up is the high prevalence of topic 4 in the school of Aristotle. Topic 3 seems quite prevalent in books on Feminism, while Plato and Stoicms seems to be unrelated to most topics, say for topic 3.

PCA visualizations of schools of philosophical thought

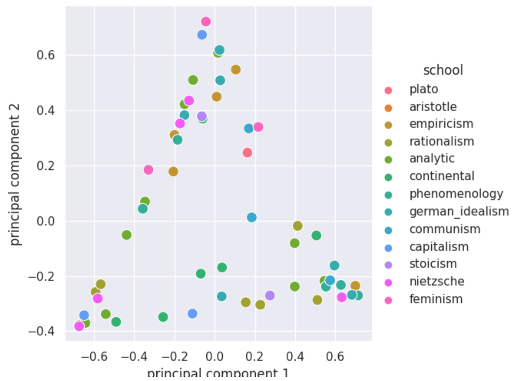

Figure 7: Title topic prevalence (projected onto a 2D PCA space). Colored by school of thought.

Figure 8: Mean topic prevalence for titles within a philosophical school (projected onto 2D PCA space)

When plotting the different book titles' intertopic distance, we can see that there seem to be some clustering going on. A lot of texts appear at the bottom left and right, as well as at the top middle. There does not seem to be any clearcut clustering of the texts from within the same philosophical school. This may lead us to believe that topic prevalence does not fully correlate with philosophical school.

When plotting the different schools' intertopic distance (grouped by mean scores of each school) interesting relations become evident. When taking the fact that the school of aristotle also seemed to be an outlier in the figure 6 (plot over the topic prevalence in schools of philosophical thought), it makes good sense.

LDA html output

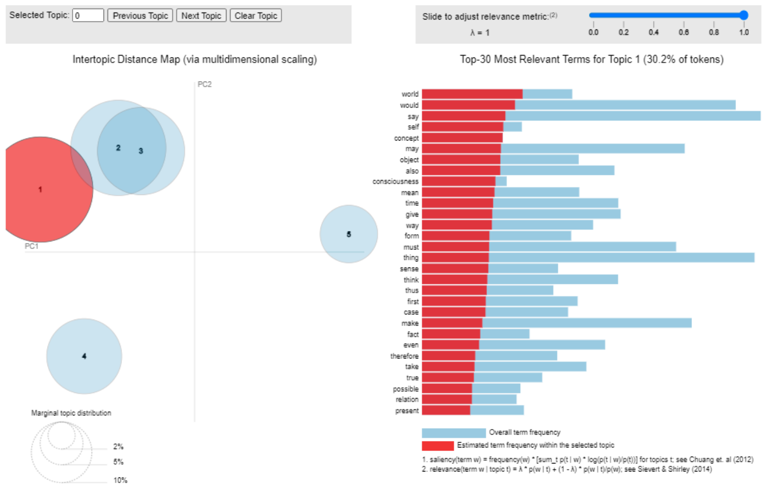

Figure 9: Screenshot of the .html file showing which words make up the topics. NOTE: The topics are ordered by presence in the data. This means that topic 1 in the .html file does not cross-correspond to topic 1 in the other visualizations.

The LDA .html output provides information on which topics are most prevalent (size of circles) and also which words are most deemed most important for the topic (words on the right). Opening the file in your favorite HTML viewer would allow you interact, by pointing your mouse to a given topic. Please note that this plot have the topics ordered by presence in the data. This means that e.g. topic 1 in the .html file does not cross-correspond to the previous plots.

4.4 Usage

Make sure to follow the instructions in the README.md located at the parent level of the repository, for the required installation of the virtual environment as well as the data download.

Subsequently, use the following code (when within the cds-language-exam folder):

Listing 6: bash terminal

```
cd assignment_5
source ../lang101/bin/activate # If not already activated
python topic_modeling_philosophy.py
```

4.4.1 Optional arguments

- ”-i”, ”–inpath”, type = str, default = os.path.join(”data”, ”philosophy_data.csv”), Default path to data required = False, help= ”str - path to image corpus”)
- ”-t”, ”–test”, type = bool, default = False, required = False, help= ”bool - specifying whether to run a test on a subset of 50000 randomly sampled entries or on the full dataset”)

5 Text classification using Deep Learning (assignment 6)

5.1 Assignment description

Winter is... hopefully over.

In class this week, we've seen how deep learning models like CNNs can be used for text classification purposes. For your assignment this week, I want you to see how successfully you can use these kind of models to classify a specific kind of cultural data - scripts from the TV series Game of Thrones.

You can find the data here.

In particular, I want you to see how accurately you can model the relationship between each season and the lines spoken. That is to say - can you predict which season a line comes from? Or to phrase that another way, is dialogue a good predictor of season?

- Start by making a baseline using a 'classical' ML solution such as CountVectorization + LogisticRegression and use this as a means of evaluating how well your model performs.
- Then you should try to come up with a solution which uses a DL model, such as the CNNs we went over in class.

5.2 Methods

Specifically for this assignment

For the Logistic Regression (LR) classification task, I start by loading in the dialogue from Game of Thrones. I then do a stratified train-test split of the data, as the data set is unbalanced. Stratification locks the distribution of classes in the train and test sets - i.e. if season 1 entries account for 23% of the entire data set, then both the train and test set will also consist of 23% data from season 1. The dialogue entries are then vectorized. The sentences are converted to vectors to allow for LR processing. Each number in the vectors represent a word index in a vocabulary list (which links the integer to the corresponding string). The feature vectors and the labels from the training data are then used to train the LR classifier. The trained model is subsequently tested on the test split for which a classification matrix and a confusion matrix is saved to the folder *out*, to assess performance.

For the Convolutional Neural Networks (CNN) classification task, the data is split in a similar fashion to the LR using stratification. The sentences are then converted to vectors as to prepare them as input for the CNN. All feature vectors are then padded with 0's until the reach the vector length of the longest vector so as to have the same length. The labels are then one-hot encoded using sklearn's LabelBinarizer() and an embedding matrix computed create using a 50D or a 100D Glove models. Which Glove model can be specified through the use of the argument *–glovedim* (for more information on the Glove models, see here). After the embedding is created, I define a sequential model using Keras and here use the embedding matrix to create an embedding layer as the first layer. The layer is then succeeded by a convolutional layer with ReLU activation, followed by a max pooling layer and two fully connected network layers. The model is then trained on the training data and tested on the test data. Using the predictions a classification matrix is saved to the folder *out*.

On a more general level (this applies to all assignments)

I have tried to as accessible and user-friendly as possible. This has been attempted by the use of:

- Smaller functions. These are intended to solve the sub-tasks of the assignment. This is meant to improve readability of the script, as well as simplifying the use of the script.
- Information prints. Information is printed to the terminal to allow the user to know what is being processed in the background.
- Argparsing. Arguments that let the user determine the behaviour and paths of the script.

5.3 Results and discussion

It is worth noting that when tuning either the C-parameter in the LR classifier or the layers in the CNN such as what has been done here, that we risk overfitting to the testing data. For this reason it is generally advisable to include a hold out set, if there is enough data for it. The hold out set functions as another test set that is only to be tested on when parameters have been tuned to maximum performance on the test set.

These scripts do not include a holdout set but have had some parameters tuned and are thus perhaps prone to overfitting. The results ought to be scrutinized accordingly.

As for assessing performance, I have chosen to focus on F1-scores (harmonic mean between recall and precision). Although accuracy is often in the focus when evaluating machine learning classification performance, it is a metric that is often misinterpreted. Using accuracy as a metric for the combined classification performance in an unbalanced data set such as this would not accurately convey the performance of the model, as classes with more data would be weighed as more important when taking the accuracy score at face value.

Logistic Regression classification

	Season 1	Season 2	Season 3	Season 4	Season 5	Season 6	Season 7	Season 8	accuracy	macro avg	weighted avg
precision	0.29	0.27	0.24	0.26	0.25	0.33	0.39	0.25	0.28	0.28	0.28
recall	0.31	0.37	0.26	0.29	0.24	0.24	0.29	0.09	0.28	0.26	0.28
f1-score	0.30	0.31	0.25	0.27	0.25	0.28	0.33	0.13	0.28	0.27	0.27
support	477.0	587.0	536.0	517.0	455.0	429.0	366.0	220.0	0.28	3587.0	3587.0

Table 3: Classification report for the LR model

The LR model performs with an average macro F1-score of 0.27 and seems to predict some seasons better than others. The F1-score for season 8 is at 0.13, which shows that the model had difficulties with predictions here. When looking at the precision and recall it becomes apparent that the model misplaced many season 8 quotes as belonging to other seasons.On the contrary, of the few that were actually classified as season 8 most did actually come from season 8. The support score for this season is significantly lower than for the other classes. This all points in the direction that the model learned that it achieved a higher performance by classifying only the quotes that it was very confident belonged to season 8, as season 8. This is due to the fact that the data set contained very few quotes from this 8.

Convolutional Neural Network classification

	Season 1	Season 2	Season 3	Season 4	Season 5	Season 6	Season 7	Season 8	accuracy	macro avg	weighted avg
precision	0.33	0.26	0.23	0.18	0.16	0.22	0.22	0.18	0.22	0.22	0.23
recall	0.34	0.27	0.14	0.36	0.13	0.08	0.32	0.09	0.22	0.21	0.22
f1-score	0.33	0.26	0.17	0.24	0.14	0.12	0.26	0.12	0.22	0.21	0.21
support	477.0	587.0	536.0	517.0	455.0	429.0	366.0	220.0	0.22	3587.0	3587.0

Table 4: Classification report for the CNN model

The CNN model performs notably worse than the LR model with a macro average F1-score of 0.21. In general deep neural networks tend to perform better that logistic regressions, given optimal layer architecture, enough data and enough training. However, in this case we see the LR model outperforming the CNN model. Why might this be? Utilizing the embedding that only excludes certain word types result in the model not being able to learn patterns that might be prevalent in these word types. It might be that the words excluded carried important information for distinguishing between seasons.

Another reason for the low performance may be related to overfitting of the model. When looking at the training history, it becomes evident that validation and training accuracy started to diverge after only a few epochs. If the model architecture had included either a dropout layer or some regularization, the overfitting may have been less of an issue.

5.4 Usage

Make sure to follow the instructions in the README.md located at the parent level of the repository, for the required installation of the virtual environment as well as the data download.

Subsequently, use the following code (when within the cds-language-exam folder):

Listing 7: bash terminal

```bash
cd assignment_6
source ../lang101/bin/activate # If not already activated
python lr_got.py
python cnn_got.py
```

5.4.1 Optional arguments

lr_got.py arguments for commandline to consider:

- "-i", "–inpath", type = str, default = os.path.join("data", "Game_of_Thrones_Script.csv"), required = False, help = "str - specifying inpath to Game of Thrones script")
- "-C", "–C", type = int, default = 1, required = False, help = "int - specifying c parameter for the model")

cnn_got.py arguments for commandline to consider:

- "-i", "–inpath", type = str, default = os.path.join("data", "Game_of_Thrones_Script.csv"), required = False, help= "str - specifying inpath to Game of Thrones script")
- "-e", "–epoch", type = int, default = 10, required = False, help= "int - specifying number of epochs for the cnn model training")
- "-b", "–batchsize", type = int, default = 100, required = False, help = "int - specifying batch size")
- "-g", "–glovedim", type = int, default = 50, required = False, help= "int - specifying which how many dimensions should be in the glove embedding to use. Options: 50 or 100")
- "-E", "–embeddingdim", type = int, default = 50, required = False, help= "int - specifying dimensions for the embedding")

6 LSTM models for text generation (assignment 7 - self-assigned)

6.1 Assignment description

This assignment seeks to investigate an approach to generate new textual content. It implements a Recurrent Neural Network (RNN) to learn word sequence patterns. Using the trained model new word sequences are generated.

More specifically, this assignment seeks see whether it is possible to generate new textual content in line with the text from the corpus of folklore fairy tales written by the Brothers Grimm (Jacob Ludwig Karl Grimm and Wilhelm Carl Grimm). The project intends to investigate the questions: *How well can a neural network learn the patterns of the writings of the Brothers Grimm?* and *Using the trained model - is it possible to generate new textual content that could have been something you might read in an old fairy tale?*

Try using a text generative approach that learns and predicts on word-level, rather than on a character-level.

- Train a Long Short-Term Memory (LTSM) artificial Recurrent Neural Network (RNN) on the corpus
- Use sequences of 50 words as input for the model
- Generate new sequences of texts, using the trained model
- Bonus task: include arguments that let you specify model training parameters and LTSM model architecture (LTSM layers, epochs and batch size). You might also want to specify options for the text generation - how many text chunks should be generated?

6.2 Methods

Specifically for this assignment

For this assignment I started out by loading in the text corpus. The text was then preprocessed; the strings that each contained a fairy tale were combined into a single string, which was divided up into a list of words. The preprocessing also entailed the removement of non-alphanumeric characters which meant that line breaks, punctuation and quotations marks were removed. I then defined and made use of a function which retrieves word sequences using a moving window, this function took window-size and step-size as arguments. To give an example: Given window-size = 3 and step-size = 1, it would retrieve the sequences [[”once upon a”], [”upon a time”], [”a time in”]] from the text [”once”, ”upon”, ”a”, ”time”, ”in”]. For this analysis, the sequences retrieved from the Grimms’ fairy tales would be of length 51 and would move by a single word at a time. The sequences were then tokenized meaning that the list of words was converted into a vector. The integers of the vector were unique for each word and functioned as an ID in a saved vocabulary list, enabling the vectors to be converted back into words. Having the text sequences as vectors allow the model in training on the data. The first 50 words in each tokenized sequence would be used as features for the model input, while the last word in the sequence would be what the model tries to predict. Before training, the tokens to be predicted were one-hot encoded, to allow for the model to use categorical crossentropy as loss function and softmax activation for the final layer. This way, the model would return an array of the probability of each of the possible predictions. The word with the highest probability would then be used as the prediction of the model.

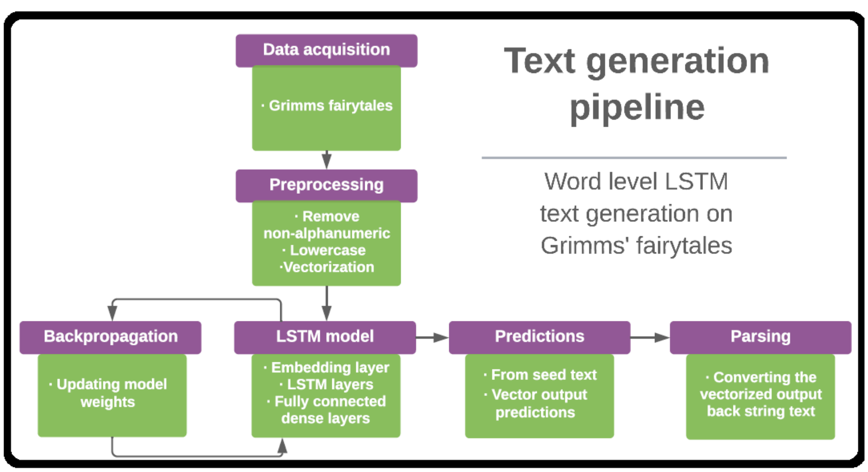

Figure 10: Visualization of the pipeline used for text generation

The model consists of an embedding layer followed by two LTSM layers of depth 128 and 100 as per default. Using the argument *–ltsmlayers*, one may specify another structure for the LTSM layers. The model implements LTSM layers due to their way of handling the vanishing gradient problem (the problem of shrinking gradients over time in backpropagation) that is prevalent in traditional RNNs. It does so by the use of feedback connections called gates. The LTSM layers are succeeded by a dense layer of 32 nodes and an output layer with a softmax activation function with number of nodes equal to number of possible predictions (number of unique words in the corpus). During model training, the model adjusts its' weights in order to learn the patterns in the sequences. This enables it in predicting the next token that would appear after each sequence of 50 tokens. Epochs and batchsize for the model training can be determined using the arguments *–epochs* and *–batchsize* (see section "Optional arguments" for more information).

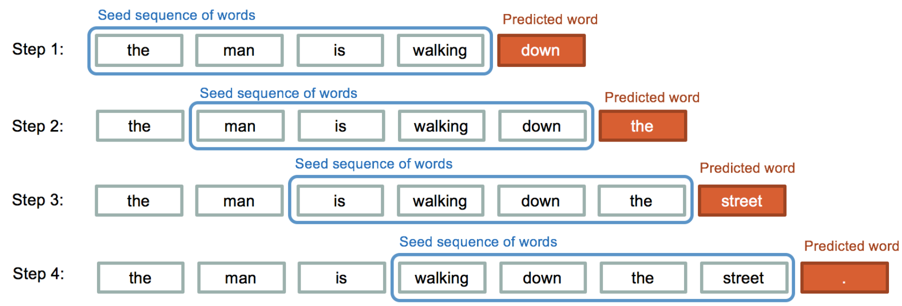

Figure 11: Visualization of the principle behind text generation algorithms. Image from blogpost by Harsh Basnal

How can a model that classifies the next word in a sequence be used to generate new text? A visualization of text generation approach used in this script can be seen above in figure 11. First it takes a sequence of words (a seed-sequence) and predict which word is the most likely to succeed. The model then uses all words in the seed-sequence except the first, plus the newly predicted word to predict another new word. Followed by a new prediction of the seed-sequence except the first two first, plus the two newly predicted words. This continues until the model has predicted and thus generated a new sequence of some specified length. It must be mentioned that the model does not do this with words, but rather with the integer values as the strings have been vectorized. Upon the completion of generating a new sequence in vector format, the tokens that all carry a unique ID are converted back into words using the saved dictionary of integer ID to word.

The seed-sequences used to generate text in this script are the same sequences that were used to train the model. When generating a new sequence the script samples from these sequences random. The number of generated sequences can be specified using the argument *–ngenerate*.

On a more general level (this applies to all assignments)

I have tried to as accessible and user-friendly as possible. This has been attempted by the use of:

- Smaller functions. These are intended to solve the sub-tasks of the assignment. This is meant to improve readability of the script, as well as simplifying the use of the script.
- Information prints. Information is printed to the terminal to allow the user to know what is being processed in the background.
- Argparsing. Arguments that let the user determine the behaviour and paths of the script.

6.3 Results and discussion

Model training

The model achieved a training accuracy of 64% which means that more than half of the word predictions were correct. Given the +2700 classes (unique words, possible to predict) it can be said that the model performed quite well. Needless to say, this is certainly because the model overfit the the data set. It learned patterns that are specific to these texts, rather than patterns that would apply to all texts. If tested on out-of-sample texts, the performance would evidently drop significantly as a result of the low generalizability. However, the purpose of this model was not generalize to other texts and to achieve high out-of-sample performance, but rather to generate texts. But did the model train enough? From looking at the plot over training accuracy and loss over epochs, it appears that the model had diminishing returns as epochs increased and more epochs would likely not have increase performance much. Given a more complex model architecture, the ceiling effect of performance may have occurred later. If such a model had been trained for more epochs, the models may have been able to perform even better.

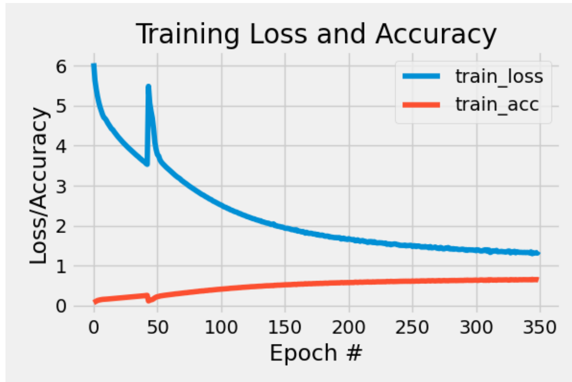

Figure 12: Training history of the model

Generating text

For clarificational purposes, I will go through two examples of generated sequences for this section. The full output of generated sequences can be found in *assignment_7/out/generated_sequences.csv* and an excerpt of this table can be found at the bottom of this section.

After manually altering two generated sequences that the model ended up producing, here are some examples of the output:

> [...] He took them a powerful gold, lying on the floor and nailed them free.
> The fifth in day, she went to bed then he went into the kitchen and said to him:
> "Can you light nothing but set me free". Wash it sleeping and did not believe that it might soon as it were. [...]

Figure 13: Example 1 of generated sequence - manually formatted

> [...] "The sun soon wanted to drink.
> The door was a poor pity and he got up into the room and wanted to have", said the king.
> As he came towards it and sat down on her head and did not fly about,
> the wolf knocked into the water and kill her. [...]

Figure 14: Example 2 of generated sequence - manually formatted

Note that these have been manually altered by adding line breaks, punctuation and by capitalizing letters after periods. The sentences seem to apply to some rules of grammar; verbs seem to be predicted in the context of nouns. Determiners such as "the" seems to accurately precede nouns, while prepositions such as "on" and "to" are correctly placed in sentences such as "*lying on the floor*" and "*she went to bed*". Although the generated content had some merits in terms of grammatical structure, there are also pitfalls. The model seems to arbitrarily guess whether a word should appear in its past of present tense - "*the wolf knocked into the water an kill her*". Furthermore almost all semantic coherence seems to be absent. A sentence such as "*The sun soon wanted to drink.*" does not make much sense. Not even when read in the context of fairy tales which is what the text is meant to resemble. Lack of semantic coherence seems to be a general issue across the different methods used to generate new text - even for esteemed experts in RNNs such as the team behind TensorFlow (see their approach here). At present, text generative processes seem to be mostly useful for entertainment purposes, generating abstract poetry or as a means to acquire inspirational content in an atypical way. When looking at the raw output of the script it also becomes evident that this model lacks the formatting that was manually applied in the two previous examples - things such as line breaks, punctuation and capitalization of letters after periods. Take a look at the examples in their unformatted raw version below.

> [...] he took them a powerful gold lying on the floor and nailed them free
> the fifth in day she went to bed then he went into the kitchen and said to him
> can you light nothing but set me free wash it sleeping and did not believe that it might soon as it were [...]

Figure 15: Example 1 of generated sequence - raw

> [...] the sun soon wanted to drink
> the door was a poor pity and he got up into the room and wanted to have said the king
> as he came towards it and sat down on her head and did not fly about
> the wolf knocked into the water and kill her [...]

Figure 16: Example 2 of generated sequence - raw

The results leave us with some information to answer the two questions posed in the description of the project: "*How well can a neural network learn the patterns of the writings of the Brothers Grimm*?" and "*Using the trained model - is it possible to generate new textual content that could have been something you read in an old fairy tale?*". It seems that this specific neural network can learn some patterns of fairy tales and of language in general. Given that the model has had no hard-coded rules implemented it can be thought impressive that it was able to produce sequences of text with a least some grammatical structure. However, it is clear that the model is not able to generate new textual content that one might have read in a fairy tale from the 1810s, as there seems to be little to no meaning in the produced texts. Moreover, the preprocessing of the data filtered away non-alphanumeric characters and this may have been unnecessary. Had, for instance, periods been treated like tokens just as the words, the model may had been able to predict punctuation somewhat accurately. Pair this with line breaks and quotation marks etc., and the output produced may have resembled text in fairy tales slightly more accurately. Other measures might also have been taken, such as applying Regex patterning to capitalize the first letter following a period.

	length_10	length_50
0	from whence his wealth came to see them to the	her own child will for this then the king went shivering and looked up and saw that their hair her sister loved his bread she thought each man caused found
1	cried is himself give me my head then the king	brought two far in the window and the man came with pale and diamonds equipped a thousand folk this cages loudly and as he saw a white snake lying coiled
2	kitchen and shook out the willowwren flew up to his	has supported me down into the cellar and see where my father grieved when we to eat of the sexton came to the door he leapt into the inn there
3	foot and limped hobblety jib hobblety and when he came	him joy at once the king had a great thirst and mother so she sat down by the fields and then he ran back to her grave and said what
4	and when he had gone on her she walked out	dead an old fox too for she began to bewail them began with gold and sleep dogs yet fiddle and then her mother should find him as he could even
5	place and when he saw the bridegroom screamed out and	after something but that you can have my thirst but happen to said the youth and was travelling dead cup to the bottom of the movements door remained a raven

Table 5: Excerpt of the full output

6.4 Usage

Make sure to follow the instructions in the README.md located at the parent level of the repository, for the required installation of the virtual environment as well as the data download.

Subsequently, use the following code (when within the cds-language-exam folder):

Listing 8: bash terminal

```bash
cd assignment_7
source ../lang101/bin/activate # If not already activated
python text_generator.py
```

6.4.1 Optional arguments

- ”-i”, ”–inpath”, type = str, default = os.path.join(”data”, ”grimms_fairytales.csv”), required = False, help = ”str - specifying inpath to the Grimms fairy tales”)
- ”-l”, ”–ltsmlayers”, type = int, nargs=’+’, default = [128, 100], required = False, help = ”list of integers - specifying number and depth of LTSM layers. e.g. –ltsmlayers 32, 64, 32”)
- ”-b”, ”–batchsize”, type = int, default = 64, required = False, help = ”int - specifying batch size for the model training”)
- ”-e”, ”–epochs”, type = int, default = 350, required = False, help = ”int - specifying number of epochs for the training”)
- ”-n”, ”–ngenerate”, type = int, default = 50, required = False, help = ”int - specifying how many sequences the script should generate”)

7 Acknowledgements

I want to thank our instructors, Ross Deans Kristensen-McLachlan and Kristoffer Laigaard Nielbo for, in spite of a pandemic, managing to put together an awesome course, which was as well-structured as it was friendly fun an educational.