Language Analytics Portfolio

Malte Højmark-Bertelsen
201809226@post.au.dk

Cultural Data Science
Aarhus University
2021

Table of Contents

Assignment 2 - String processing with Python

GitHub link for general repository:
https://github.com/MalteHB/language_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/language_analytics_cds/blob/main/src/collocation.py

Project Description

Description for assignment 2:

"String processing with Python
*Using a text corpus found on the cds-language GitHub repo **or** a corpus of your own found on a site such as Kaggle, write a Python script which calculates **collocates** for a specific keyword.*

- *The script should take a directory of text files, a keyword, and a window size (number of words) as input parameters, and an output file called out/{filename}.csv*
- *These parameters can be defined in the script itself*
- *Find out how often each word collocates with the target across the corpus*
- *Use this to calculate mutual information between the target word and all collocates across the corpus*
- *Save result as a single file consisting of three columns: collocate, raw_frequency, MI*
- *BONUS CHALLENGE: Use argparse to take inputs from the command line as parameters*

General instructions
- *For this assignment, you should upload a standalone .py script which can be executed from the command line.*
- *Save your script as collocation.py*
- *Make sure to include a requirements.txt file and your data*
- *You can either upload the scripts here or push to GitHub and include a link - or both!*
- *Your code should be clearly documented in a way that allows others to easily follow the structure of your script and to use them from the command line*

Purpose

This assignment is designed to test that you have a understanding of:

1. *how to structure, document, and share a Python scripts;*
2. *how to effectively make use of native Python packages for string processing;*
3. *how to extract basic linguistic information from large quantities of text, specifically in relation to a specific target keyword"*

Methods

To calculate collocates I firstly needed a text corpus. I chose to use the '*100 English Novels'* corpus available on the cds-language GitHub. It can also be found on the GitHub repository for this entire project-collection of assignments in the folder '*data*'. To calculate the collocates of a given target word I started by loading the specified number of novels, or files, from the corpus, into a list, and afterwards tokenizing the files by whitespace. After this I iterated through the tokens and collected the collocates within a specified window size. For this specific experiment I chose the word "*world"* as the target word with a windows size of 2. Calculating the raw frequency for each collocate was done in a simple for-loop with a counter, however, the MI-score was slightly more difficult. This was done by firstly, calculating the joint frequency, O11, with the target word as keyword and each collocate as collocate within the specified window size. Then I calculated the disjoint frequency with the target word as keyword and each collocate as collocate, O12, within the specified window size and summed these frequencies O11 + O12 = R1. Afterwards I calculated the disjoint frequency with each collocate as keyword and the target word as collocate, O21, within the specified window size and summed this frequency with the joint frequency, O11 + O21 = C1. Subsequently, I calculated the expected frequency by multiplying R1 and C1 and dividing these with the number of tokens, and finally took the log, by using the '*NumPy'* package (Harris et al., 2020), of O11 divided by E11, finally resulting in the MI-score. All the collocates, raw frequencies and MI-scores were finally put into a dataframe using the '*pandas'* package (McKinney, 2010; team, 2020).

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/language_analytics_cds.git
cd language_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
```

```
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

Finally to run the scripts, run the following lines:

```
# Run the collocation script and to limit the amount of time use the
--number_of_files argument. I recommend 5 files for a proof of concept
python src/collocation.py --number_of_files 5


# Optionally you can use the -h flag to see all the modifiable arguments:
python src/collocation.py -h
# Output:
usage: collocation.py [-h] [--dd Data Directory] [--od Output Directory] [--kw
Keyword] [--ws Window Size]
                      [--number_of_files Number of Files]

optional arguments:
  -h, --help            show this help message and exit
  --dd Data Directory   A PosixPath to the data directory.
  --od Output Directory
                        A path to the output directory.
  --kw Keyword          The keyword to find collocates for.
  --ws Window Size      The given window size.
  --number_of_files Number of Files
                        Number of files to create collocates from in the data
directory.
```

Discussion of Results

Running the script with the target word set to '*world'*, a windows size of 2 and only using five files produced the dataframe 'collocates_5_files.csv' which can be accessed in the 'out' folder of the repository. It is sorted by raw frequency, and the first 5 rows can be seen in table X. Here it seems that the most frequent words in the corpus are also some of the most frequently used words in the natural language. These words are also referred to as stop words, and I would, thus, recommend for further development to include a stop word removal prior to finding collocates, if one wanted to mitigate this issue.

Target Word: "*world*"	**Collocate**	**Raw Frequency**	**MI**
	the	27454	4.009379
	and	16412	2.105773
	to	14221	2.090311
	of	13789	2.871684
	I	11522	1.656775

Table 1: First five rows of the resulting dataframe.

Assignment 3 - Sentiment analysis

GitHub link for general repository:
https://github.com/MalteHB/language_analytics_cds

GitHub link for the specific python scripts:
https://github.com/MalteHB/language_analytics_cds/blob/main/src/sentiment.py

Project Description

Description for assignment 3:

*"**Dictionary-based sentiment analysis with Python***

Download the following CSV file from Kaggle:

https://www.kaggle.com/therohk/million-headlines

*This is a dataset of over a million headlines taken from the Australian news source ABC (Start Date: **2003-02-19** ; End Date: **2020-12-31**).*

- *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?*

- ***HINT:** You'll probably want to calculate an average score for each day first, before calculating the rolling averages for weeks and months.*

General instructions

- *For this assignment, you should upload a standalone .py script which can be executed from the command line **or** a Jupyter Notebook*
- *Save your script as sentiment.py **or** sentiment.ipynb*

- *Make sure to include a requirements.txt file and details about where to find the data*
- *You can either upload the scripts here or push to GitHub and include a link - or both!*
- *Your code should be clearly documented in a way that allows others to easily follow the structure of your script and to use them from the command line*

Purpose
This assignment is designed to test that you have a understanding of:

1. *how to perform dictionary-based sentiment analysis in Python;*
2. *how to effectively use pandas and spaCy in a simple NLP workflow;*
3. *how to present results visually, working with datetime formats to show trends over time"*

Methods

To conduct the sentiment analysis of the headlines i used the library '*spaCy'* (Honnibal et al., 2020) and specifically the pipeline spaCy TextBlob which is built upon the package with the similar name, '*TextBlob'* (Loria, 2018). Here I created an NLP pipeline and calculated the sentiment score for each headline in the text corpus, and adding this sentiment score to a list. The list was then added to a pandas dataframe (McKinney, 2010; team, 2020), and here I could create a rolling average set to both seven days, weekly. and 30 days, monthly. One obvious limitation of doing this is that not all months are exactly 30 days long, and thus, should further work investigate other rolling average calculation options. Finally two images of the rolling averages was created using the Python package '*Matplotlib'* (Hunter, 2007).

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/language_analytics_cds.git
cd language_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

Since this assignment was done using the spaCy library it is also necessary to download an english pretrained language model from spaCy:

```
# Download the english language model from spacy
python -m spacy download en_core_web_sm
```

Finally to run the scripts, run the following lines:

```
# Run the sentiment script
# If you want to, you can reduce runtime by setting the argument --n_headlines,
which specifies how many headlines from the top of the .csv file you want to
load.

# For a all the headlines (and thereby a long runtime):
python src/sentiment.py

# For a proof of concept and a decreased runtime:
python src/sentiment.py --n_headlines 100000

# Optionally you can use the -h flag to see all the modifiable arguments:
python src/sentiment.py -h
# Output:
usage: sentiment.py [-h] [--dd Data Directory] [--od Output Directory]
[--n_headlines Number of headlines] [--bs Batch Size]

optional arguments:
  -h, --help            show this help message and exit
  --dd Data Directory   A PosixPath to the data directory.
  --od Output Directory
                        A path to the output directory.
  --n_headlines Number of headlines
                        The number of headlines to load from a dataframe
  --bs Batch Size       The size of each batch processed by the spaCy pipeline
```

Discussion of Results

Running the script with the approximately one million headlines resulted in the two plots seen in image 1 and image 2. For the image with the weekly rolling average it seems that there is a slight decrease in polarity towards '08-'09 which could be due to the financial crisis. From there,

however, it just goes slightly up, but further statistical tools are needed to assess the significance of this increase. It does however, seem that the polarity of the headlines lean slightly towards the positive side of things, rather than the negative side as the tendency is marginally above 0.00. The same tendencies can be drawn from the monthly averaging, where the biggest difference between the two averaging methods seems to be the spread of polarity. For both it also seems that in the beginning of 2020, the headlines started becoming more positive. One could, thus, speculate that Australia had not yet entered the global COVID-19 pandemic…

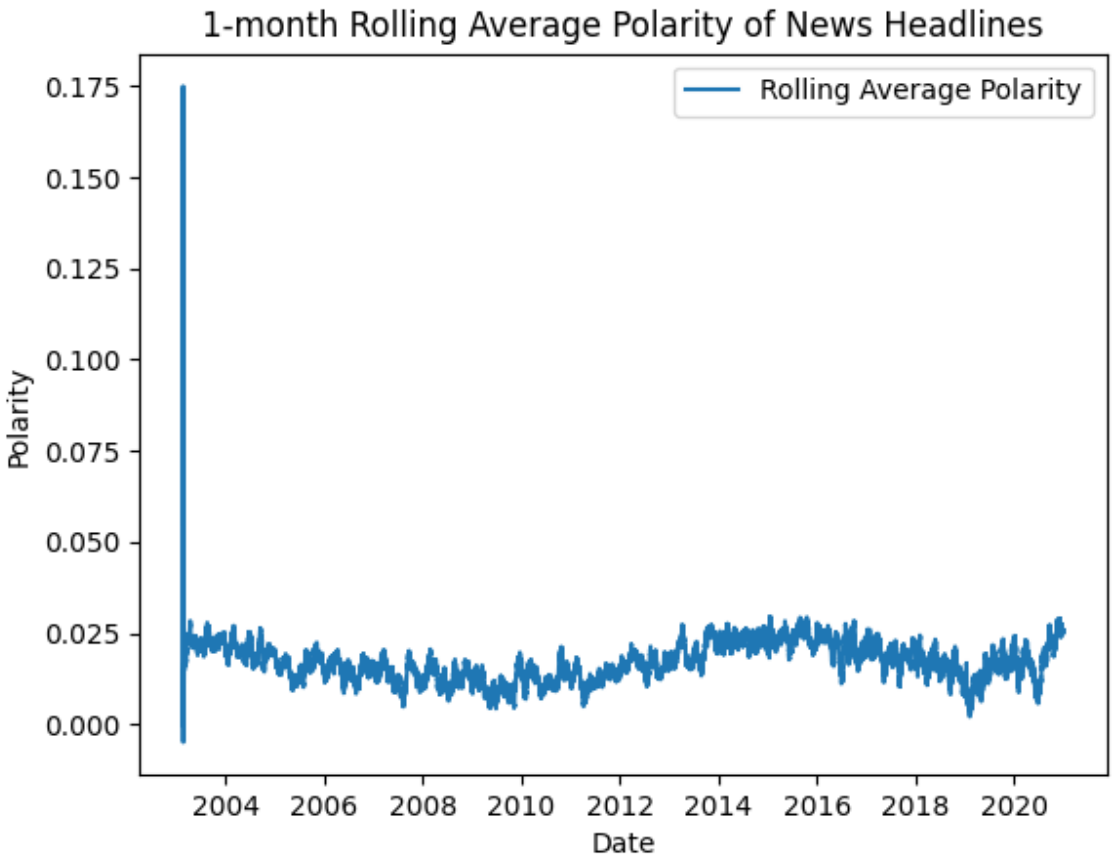

Image 1: 1-month rolling average polarity of news headlines.

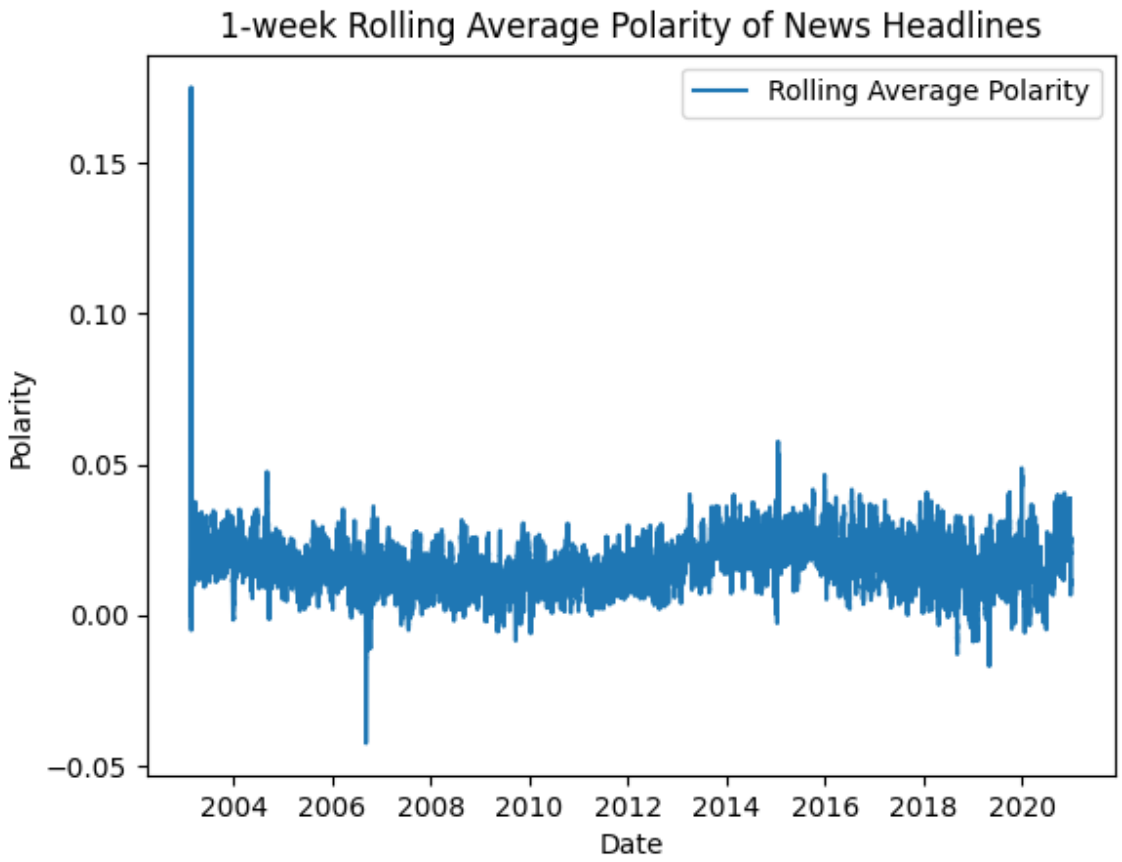

Image 2: 1-week rolling average polarity of news headlines.

Assignment 4 - Network analysis

GitHub link for general repository:
https://github.com/MalteHB/language_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/language_analytics_cds/blob/main/src/network.py

Project Description

Description for assignment 4:

"**Creating reusable network analysis pipeline**

This exercise is building directly on the work we did in class. I want you to take the code we developed together and in you groups and turn it into a reusable command-line tool. You can see the code from class here:

https://github.com/CDS-AU-DK/cds-language/blob/main/notebooks/session6.ipynb

This command-line tool will take a given dataset 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***.

Tips

- You should use **argparse()** in the Python standard library
- Your code should contain a **main()** function
- Don't worry too much about efficiency - networkx is really slow, there's no way around i!
- If you have issues with **pygraphviz**, just use the built-in matplotlib functions in networkx.

- You may want to create an argument for the user to define a cut-off point to filter data. E.g. only include node pairs with more than a certain edge weight.
- Make sure to use all of the Python scripting skills you've learned so far, including in the workshops with Kristoffer Nielbo

Bonus challenges

- Attempt to implement coreference resolution on entities (time-consuming)
- Bundle your code up into a Python class, focusing on code modularity
- Let the user define which graphing algorithm they use (pretty tricky)
- Are there other ways of creating networks, rather than just document co-occurrence? (really tricky)

General instructions

- For this assignment, you should upload a standalone *.py* script which can be executed from the command line
- Save your script as *network.py*
- You must include a *requirements.txt* file and a bash script to set up a virtual environment for the project. You can use those on worker02 as a template
- You can either upload the scripts here or push to GitHub and include a link - or both!
- Your code should be clearly documented in a way that allows others to easily follow the structure of your script and to use them from the command line

Purpose

This assignment is designed to test that you have an understanding of:

1. how to create command-line tools with Python;
2. how to perform network analysis using networkx;
3. how to create resuable and reproducible pipelines for data analysis.*"*

Methods

To create the command-line tool, I created a Python-script in which the class 'NetworkAnalysis' houses the capability of conducting, as the name implies, network analysis. For this project a network analysis was done on the weighted edgelist file, called *'edges_df.csv'* located in the data folder of this project repository, which was created in class. The network analysis was done by first filtering out the edges below the minimum edge weight. Afterwards the package *'NetworkX'* (Hagberg et al., 2008) was used to create a network graph with the Fruchterman-Reingold force-directed algorithm (Fruchterman & Reingold, 1991), which can be employed by using the networkx.spring_layout() function. The graph was saved as an image

using the Python library 'Matplotlib' (Hunter, 2007), and finally, the degree, betweenness and eigenvector centrality measures were calculated.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/language_analytics_cds.git
cd language_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

Finally to run the scripts, run the following lines:

```
# Run the network analysis script
python src/network.py

# Optionally you can use the -h flag to see all the modifiable arguments:
python src/network.py -h
# Output:
usage: network.py [-h] [--i Input File] [--god Graph Output Directory] [--sg
Save graph] [--dad Data Output Directory] [--mew Minimum Edge Weight] [--k
Optimal distance between nodes] [--sdf Save Dataframe]

optional arguments:
  -h, --help            show this help message and exit
  --i Input File        A path to the input file.
  --god Graph Output Directory
                        A path to the graph output directory.
  --sg Save graph       Whether to save a plot of the graph. Defaults to True.
  --dad Data Output Directory
                        A path to the data output directory.
  --mew Minimum Edge Weight
                        The minimum edge weight for the network nodes. Defaults
```

```
to 1000.
  --k Optimal distance between nodes
                        Optimal distance between nodes. Increase to create more
separation and decrease to create less. Defaults to 5.
  --sdf Save Dataframe  Whether to save the dataframe of centrality measures.
Defaults to True.
```

Discussion of Results

The resulting visualized network can be seen in image X, while the first five rows of the centrality measures can be seen in table X. For a view of the full dataframe see the file '*centrality_df.csv'* in the folder '*out'* in the GitHub repository of this project. Both the visualization of the network and the centrality measures shows that the node "*Clinton*" is connected with a lot of nodes. Furthermore, it seems that there is both a "*Clinton"*and a "*Hillary Clinton",* meaning that the entities could have been counted twice in the creation of the weighted edgelist, thus leading to a more prominent centrality.

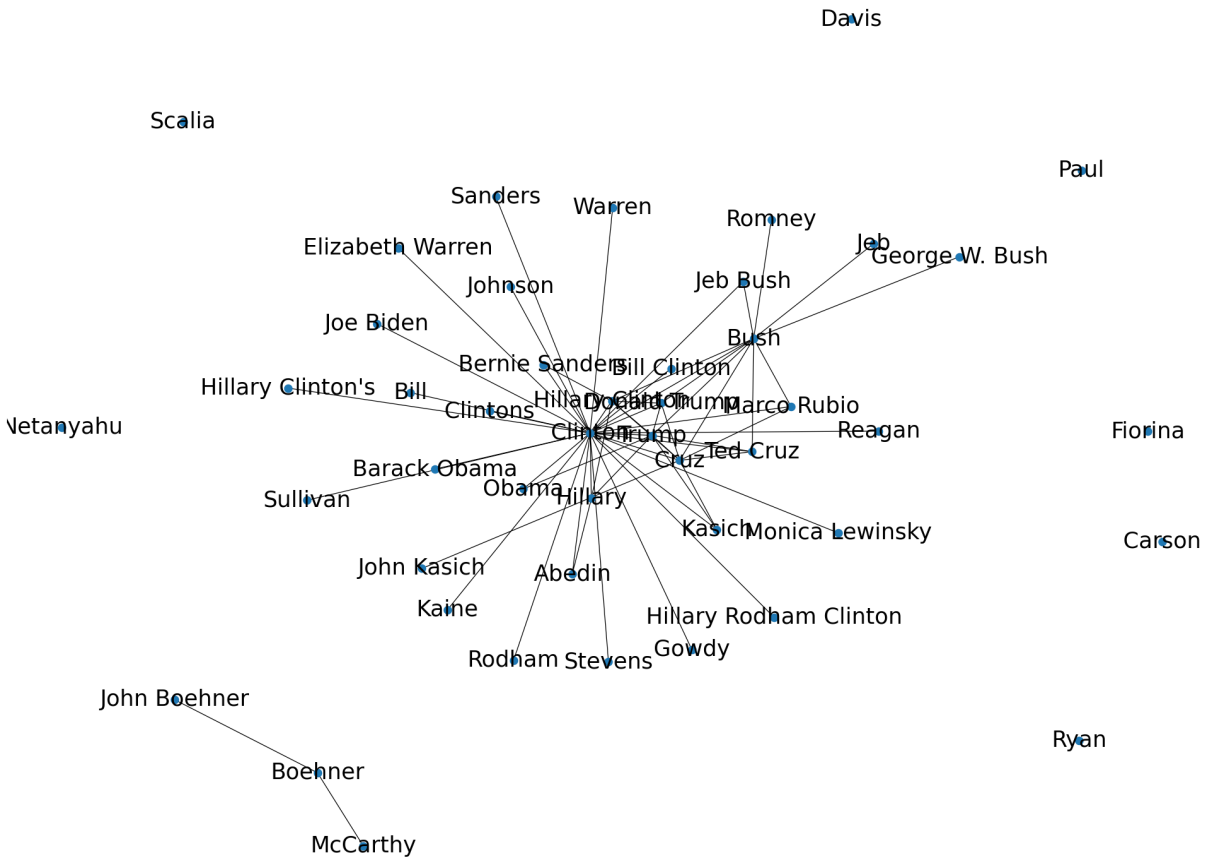

Image 3: The network created by using the Fruchterman-Reingold force-directed algorithm.

Centrality Measures	Nodes	Degree	Betweennes s	Eigenvector
	Clinton	0.73	0.51	0.54
	Bush	0.29	0.11	0.33
	Cruz	0.24	0.04	0.32
	Trump	0.24	0.01	0.33
	Hillary Clinton	0.2	0.01	0.29

Table 2: First five rows of the resulting dataframe.

Assignment 5 - (Un)supervised machine learning

GitHub link for general repository:
https://github.com/MalteHB/language_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/language_analytics_cds/blob/main/src/HF_token_classification.py

Project Description

Description for assignment 5:
"**Applying (un)supervised machine learning to text data**

The assignment this week involves a little bit of a change of pace and a slightly different format. As we have the Easter break next week, you also have a longer period assigned to complete the work.

For this task, *you will pick your own dataset to study*.

This dataset might be something to do with COVID-19 discourse on Reddit; IMDB reviews; newspaper headlines; whatever it is that catches your eye. However, I strongly recommend using a text dataset from somewhere like Kaggle - https://www.kaggle.com/datasets

When you've chosen the data, do **one** of the following tasks. One of them is a supervised learning task; the other is unsupervised.

EITHER

- Train a **text classifier** on your data to predict some label found in the metadata. For example, maybe you want to use this data to see if you can predict sentiment label based on text content.

OR

- 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 dataset.

You should formulate a short research statement explaining why you have chosen this dataset 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 dataset because I am interested in... I wanted to see if it was possible to predict X for this corpus.*

In this case, your peer reviewer will not just be looking to the quality of your code. Instead, they'll also consider the whole project including choice of data, methods, and output. Think about how you want your output to look. Should there be visualizations? CSVs?

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

Bonus challenges

- Do both tasks - either with the same or different datasets

General instructions

- You should upload standalone *.py* script(s) which can be executed from the command line
- You must include a *requirements.txt* file and a bash script to set up a virtual environment for the project You can use those on worker02 as a template
- You can either upload the scripts here or push to GitHub and include a link - or both!
- Your code should be clearly documented in a way that allows others to easily follow the structure of your script and to use them from the command line

Purpose

This assignment is designed to test that you have an understanding of:

1. how to formulate research projects with computational elements;
2. how to perform (un)supervised machine learning on text data;
3. how to present results in an accessible manner."

Methods

For the fifth assignment of the Language Analytics course a short research project was tasked. The research project I chose was to investigate a possible framework to use for text classification. Specifically I wanted to investigate the fine-tuning capabilities of the Hugging Face (Wolf, Debut, Sanh, Chaumond, Delangue, Moi, Cistac, Rault, Louf, Funtowicz, & Brew, 2020) Python library '*Transformers'* (Wolf, Debut, Sanh, Chaumond, Delangue, Moi, Cistac, Rault, Louf, Funtowicz, Davison, et al., 2020), and specifically the '*Trainer()*' method in a token classification task. The reason for this was that the Trainer()-method automatizes a lot of the configuration setup for machine learning, including automatized hyperparameter optimization.

Furthermore, I wanted to do it for Danish, since Danish is currently in a technological disadvantageous position compared to bigger languages such as English (Kirkedal et al., 2019). For Danish token classification there is, to my knowledge, only one dataset, namely DaNE (Hvingelby et al., 2020). This dataset follows the CoNLL-2003 annotation scheme and is, therefore, utilizable for several token classification tasks. The token classification task I chose to focus on was Named Entity Recognition (NER).

To do this is created the script 'HF_token_classification.py' which houses the class '*HuggingFaceTokenClassification'*. The script supports any Danish or English token classification Hugging Face dataset as well as any Danish or English pretrained language model available from Hugging Face. By default, the pretrained language model used is a small version of ELECTRA (Clark et al., 2020) (and for Danish, Ælæctra (Højmark-Bertelsen, 2021)). Running the script fine-tunes a model, which is saved to a folder called 'models' in the 'out' folder of the current repository. I also implemented a 'predict()' function, independent from Hugging Face functionality, that relies on the Python package PyTorch (Paszke et al., 2019), capable of taking a locally saved model and any given sentence, and predicting the entities in the sentence. Finally, using the '--test' flag, the model is evaluated on the test split of the DaNE dataset, and the precision, recall and F1 scores are calculated and saved to a csv in the 'out' folder. For this project the F1-scores from a fine-tuned Ælæctra model produced by the Trainer()-method is compared to metrics from other Danish NER models, obtained from the Alexandra Institute's DaNLP GitHub repository (Alexandrainst/Danlp, 2019/2021). One important limitation of this comparison is, however, that DaNLP omits the 'MISC' tag as they posit that it has limited use, and their models can thus focus more on the other available entities.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/language_analytics_cds.git
cd language_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

Finally, to run the script, run the following line:

```
# Run the Hugging Face Token Classification script
python src/HF_token_classification.py

# Optionally you can use the -h flag to see all the modifiable arguments:
python src/HF_token_classification.py -h
# Output:
usage: HF_token_classification.py [-h] [--ds Dataset] [--l Language] [--hfmp
HuggingFace Model Path] [--lmp Local Model Path]
                                  [--bs Batch Size] [--lr Learning Rate] [--e
Epochs] [--sm Save Model] [--t Task] [--s Sentence] [--test]
                                  [--p]

optional arguments:
  -h, --help            show this help message and exit
  --ds Dataset          Dataset to use from HuggingFace. Defaults to "dane".
  --l Language          Language of the dataset. Defaults to "danish".
  --hfmp HuggingFace Model Path
                        Model to use from the HuggingFace model hub.
  --lmp Local Model Path
                        Path to save the model to.
  --bs Batch Size       Batch Size to the model training setup.
  --lr Learning Rate    Learning Rate for the model training setup.
  --e Epochs            Number of epochs.
```

```
  --sm Save Model       Whether to save the model
  --t Task              What task to take from the HuggingFace Token
Classification Dataset
  --s Sentence          Sentence to be predicted by the trained model.
  --test                Test the model
  --p                   Whether to only predict a sentence.
```

For this project you can also parse the '*--p*' and '*--s*' arguments to predict an input sentence if a saved model is available:

```
# Using the '--p' and '--s' arguments:
python src/HF_token_classification.py --p --s "Malte er en dreng, som går
på Aarhus Universitet, men kommer fra Randers C"
# Output:


INITIALISING PREDICTION OF INPUT SENTENCE: 'Malte er en dreng, som går på
Aarhus Universitet, men kommer fra Randers C'


Input Tokens: Malte er en dreng , som går på Aarhus Universitet , men
kommer fra Randers C

Predicted Entities: B-PER O O O O O O O B-LOC I-ORG O O O O B-LOC I-ORG

Probabilities: [0.9467394948005676, 0.9972938895225525, 0.9972898960113525,
0.997260332107544, 0.9973267316818237, 0.9972967505455017,
0.9972966313362122, 0.9971480965614319, 0.7375102639198303,
0.6075906753540039, 0.9972272515296936, 0.9972973465919495,
0.9972959160804749, 0.9972695708274841, 0.8356436491012573,
0.48209160566329956]

DONE! Have a nice day. :-)
```

Discussion of Results

Using the Trainer()-method resulted in the slew of results seen in Table 3. Here it is evident that the Trainer()-method (average F1 = 71.98) does not manage to create novel state-of-the-art results for Danish Named Entity Recognition. However, I think it is important to note that comparing the Trainer()-method to the model of similar architecture, namely the 'NERDA electra' model (average F1 = 76.77), it only comes out 4.79 F1-scores worse. And for a

command-line script, I would deem it quite impressive. Furthermore, as mentioned, these F1-scores are not truly comparable as the Trainer()-method is also fine-tuned with the '*MISC'* entity. Future improvements of the script could thus implement the ability to choose what entity-tags to incorporate in a given fine-tuning.

Test Results	LOC	ORG	PER	AVG F1
Trainer()-method	71.59	66.46	85.52	71.98
BERT	83.90	72.98	92.82	84.04
Flair	84.82	62.95	93.15	81.78
spaCy	75.96	59.57	87.87	75.73
Polyglot	64.95	39.3	78.74	64.18
NERDA (mBERT)	80.75	65.73	92.66	80.66
NERDA (electra)	77.67	60.13	90.16	76.77
DaCy (medium) v0.0.0	83.96	66.23	90.41	80.09
DaCy (large) v0.0.0	**85.29**	**79.04**	**95.53**	**86.64**

Table 3: Test results using the Trainer()-method and results reported by DaNLP.

Self Assigned Portfolio - Danish topicformers modelling

GitHub link for general repository:
https://github.com/MalteHB/language_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/neural_style_transfer.py

Project Description

For the self assigned portfolio assignment I wanted to create a Danish topic modelling script, but I wanted to utilize Transformer-based (Vaswani et al., 2017) models. Specifically, I wanted to utilize the monolingual features of multilingual models, pre-trained with knowledge distillation (Reimers & Gurevych, 2020). Furthermore, since this is a Cultural Data Science course I wanted to create some visual output for the user, and I therefore also wanted to include the capability of creating word clouds.

Methods

To do the topic modelling I created an executable script called '*danish_topicformers.py',* which has the class '*DanishTopicFormers'*. The class leverages an ingenious Python library called '*BERTopic'* (Grootendorst, 2020), that not only uses sentence embeddings created by a Transformer-based model, but also a class-based TF-IDF (c-TF-IDF), where a set of documents from a given class, can be combined into a single document, where TF-IDF then can be applied to extract specific topics.

The pre-trained models that I chose to make available in the script are four sentence-based models, namely *'distiluse-base-multilingual-cased-v2', 'stsb-xlm-r-multilingual', 'paraphrase-xlm-r-multilingual-v1', quora-distilbert-multilingual'*. The reason for this is these are the one with Danish language capabilities from the '*sentence-transformers'* repository (Reimers & Gurevych, 2019), which is used as a backend embedding module in BERTopic. By default the model *'distiluse-base-multilingual-cased-v2'* is used.

To do the topic modelling, the default dataset is a Danish dataset consisting of danish political comments (steffan267, 2019/2021), and is available from the Hugging Face datasets package (Wolf, Lhoest, Platen, Jernite, Drame, Plu, Chaumond, Delangue, Ma, Thakur, Patil, et al., 2020).

For the word clouds I used the Python library '*wordcloud'* (Oesper et al., 2011), where I furthermore, had to create a distribution dictionary of the different topic words. To do this I calculated the softmax probability for each word. This dictionary is then used in the function *'create_wordcloud()'*. The *'create_wordcloud()' f*unction automatically writes the word clouds as

'.png' files to a folder called '*wordclouds'* in the *'out'* directory. By default the script calls the function '*create_most_frequent_wordclouds()'* which is a wrapper for *'create_wordcloud()'* that saves a word cloud for each of the nine most frequent topics.

I also wanted the script to be capable of finding the most semantically similar texts for a given topic, and I therefore created the function '*get_closest_texts()'* using NumPy (Harris et al., 2020). Once a topic model is trained, the argument '*--tn'* can be used to pass a specific topic number into the function.

Lastly, BERTopic comes with a wide variety of visualization options and one is a dimensionality reduced interactive tool, and when '*danish_topicformers.py'* is run, this visualization is saved as an .html file to the *'out'*-folder as *'interactive_topics_html'*.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/language_analytics_cds.git
cd language_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

Finally, to run the script, run the following line:

```
# Run the Danish Topicformers script
python src/danish_topicformers.py

# Optionally you can use the -h flag to see all the modifiable arguments:
python src/danish_topicformers.py -h
# Output:
usage: danish_topicformers.py [-h] [--ds Dataset] [--em Embedding Model] [--tn
Topic Number] [--od Output Directory] [--sm]
                              [--sp Model Save Path] [--lm] [--lp Model Load
Path] [--kht] [--rs]
```

```
optional arguments:
  -h, --help            show this help message and exit
  --ds Dataset          Dataset to use from HuggingFace. Defaults to "dane".
  --em Embedding Model  One of the multilingual models available from
sentence-transformers.
  --tn Topic Number     Topic number for getting the closest topics
  --od Output Directory
                        A path to the output directory.
  --sm                  Whether to save a model or not to
  --sp Model Save Path  A path to the model output directory.
  --lm                  Whether to load a model or not to
  --lp Model Load Path  A path to the model output directory.
  --kht                 Whether to keep the text of a hashtag or not to
  --rs                  Whether to remove stopwords or not to
```

Discussion of Results

From running '*danish_topicformers.py',* using the sentence-transformer model, *'distiluse-base-multilingual-cased-v2'* and a class-based TF-IDF-method on a Danish political comments dataset, a topic model was created. The nine most frequent topics can be seen in table 4 and the word clouds for the corresponding topics, can be seen in images 4-6. Here it seems that the political comments refer to a variety of topics, including danes, vaccines, ministers and their payment, brexit, politicians, retirement, voting, and perhaps paternal leave. For the most frequent topic, '*31_danmark_danske_dansk_danskere'* examples of the texts most similar to the topic can be seen in table 5, and here it seems that the only comment that does not refer to anything Danish is text number 3871, which instead addresses name changes and mentions an american study. The mention of the 'american study' could be what confuses models into thinking that this is close to the topic regarding 'Danish and danes' as 'american' and 'danish' share some similar semantic meaning.

If one wanted to enhance the topic modelling powers of the script, one could try to implement inherently Danish models, and see whether these would result in other topics or whether these could differentiate 'american' and 'Danish'.

Topic	Count	Name
31	418	31_danmark_danske_dansk_danskere
41	346	41_vaccineret_vaccinere_vaccinen_vaccine
52	270	52_minister_håber_løn_statsminister
11	257	11_brexit_england_briterne_britiske
97	170	97_hørte_bålet_mur_dette
108	157	108_politikere_politik_politikerne_politisee
109	146	109_pension_førtidspension_pensionsalder_ældre
105	145	105_afstemning_stemme_vælgere_valgkamp
79	127	79_arbejde_job_baby_arbejdspladser

Table 4: The nine most frequent topics, the number of texts in the data that addresses each topic and the name of the topic.

Text Number	Text
3791	Hvis han er dansk hvorfor hedder han så ikke Søren eller Peter?
2515	Lad nu være med at være på Twitter, danske politikere.
3871	Vi kender ikke raten af navneændringer, som ses i det amerikanske studie, så det er svært at bestemme hvor på "navneassimilationskurve
3802	Ser efternavn der ikke ser dansk ud: kan kun være en udenlandsk løgner der ikke taler sproget.
6484	...men i Corydon/Thorning/Vestager-regeringen var hun en katastrofe for Danmark
6488	Danmark er jo et svindler land hvor alle åbenbart snyder og bedrager.
3795	Hvordan kan nogen se et udenlandsk efternavn og straks gå ud fra at personen ikke er dansk?

Table 5: Examples of the most similar texts for the most frequency topic, '*31_danmark_danske_dansk_danskere'*.

Image 4: Word clouds for the topics 31, 41 and 52.

Image 5: Word clouds for the topics 11, 97 and 108.

Image 6: Word clouds for the topics 109, 105 and 79.

References

Alexandrainst/danlp. (2021). [Python]. Alexandra Institute. https://github.com/alexandrainst/danlp (Original work published 2019)

Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. *ArXiv:2003.10555 [Cs]*. http://arxiv.org/abs/2003.10555

Fruchterman, T. M., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, *21*(11), 1129–1164.

Grootendorst, M. (2020). *BERTopic: Leveraging BERT and c-TF-IDF to create easily interpretable topics.* (v0.7.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.4381785

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference* (pp. 11–15).

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Højmark-Bertelsen, M. (2021). *Ælæctra—A Step Towards More Efficient Danish Natural Language Processing*.

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). *spaCy: Industrial-strength Natural Language Processing in Python*. Zenodo. https://doi.org/10.5281/zenodo.1212303

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science &*

Engineering, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Hvingelby, R., Pauli, A. B., Barrett, M., Rosted, C., Lidegaard, L. M., & Søgaard, A. (2020). DaNE: A Named Entity Resource for Danish. *Proceedings of the 12th Language Resources and Evaluation Conference*, 4597–4604. https://www.aclweb.org/anthology/2020.lrec-1.565

Kirkedal, A., Plank, B., Derczynski, L., & Schluter, N. (2019). The Lacunae of Danish Natural Language Processing. *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, 356–362. https://www.aclweb.org/anthology/W19-6141

Loria, S. (2018). Textblob Documentation. *Release 0.15*, *2*.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Oesper, L., Merico, D., Isserlin, R., & Bader, G. D. (2011). WordCloud: A Cytoscape plugin to create a visual semantic summary of networks. *Source Code for Biology and Medicine*, *6*(1), 7.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8026–8037). Curran Associates, Inc. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. http://arxiv.org/abs/1908.10084

Reimers, N., & Gurevych, I. (2020). Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. *ArXiv:2004.09813 [Cs]*. http://arxiv.org/abs/2004.09813

steffan267. (2021). *Steffan267/Sentiment-Analysis-on-Danish-Social-Media*. https://github.com/steffan267/Sentiment-Analysis-on-Danish-Social-Media (Original work published 2019)

team, T. pandas development. (2020). *pandas-dev/pandas: Pandas* (1.1.5) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 5998–6008). Curran Associates, Inc. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv:1910.03771 [Cs]*. http://arxiv.org/abs/1910.03771

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wolf, T., Lhoest, Q., Platen, P. von, Jernite, Y., Drame, M., Plu, J., Chaumond, J., Delangue, C., Ma, C., Thakur, A., Patil, S., Davison, J., Scao, T. L., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Brandeis, S., Gugger, S., … Tordjmann, A. (2020). Datasets. *GitHub. Note: Https://Github.Com/Huggingface/Datasets*, *1*.