Cultural Data Science

Language Analytics Exam 2021

Johan Kresten Horsmans
AU ID: au618771

School of Communication and Culture,
Aarhus University
Jens Chr. Skous Vej 2, 8000 Aarhus, Denmark

May 27th, 2021

Regarding code:

This portfolio submission contains the written part of my exam for Language Analytics. For the accompanying code, please refer to the main repository at: https://github.com/JohanHorsmans/cds-language-exam-2021.

To run the assignments, you need to go through the following steps in your bash-terminal to download the needed data and configure a virtual environment with the needed prerequisites for all assignments.

NOTE: The code and scripts below will only work for Linux, OS X and Worker02:

```
cd {directory where you want the assignment saved}
git clone https://github.com/JohanHorsmans/cds-language-exam-2021.git
cd cds-language-exam-2021
bash create_lang_venv.sh
source lang_venv/bin/activate
```

If you wish to uninstall the virtual environment after running the code, you need to run the following in your bash-terminal:

```
cd {cds-language-exam-2021}
bash delete_lang_venv.sh
```

Contents

1 Assignment 2

1.1 Github link

My code for assignment 2 can be found here: https://github.com/JohanHorsmans/cds-language-exam-2021/tree/main/assignment_2

1.2 Official description from instructor

ASSIGNMENT 2: Dictionary-based sentiment analysis 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:

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

1.3 Methods

The problem in this assignment relates to manually calculating collocates for a specific keyword and window-size in a text corpus. The window size is a measure for the distance there has to be between the keyword and other words for them to be considered as collocate-pairs. For the task, I used the 100 English Novels-dataset containing a total of 100 .txt-files with novels from the 19th and 20th century.

To solve the task i started by loading- and preprocessing the text to make it lowercase and remove all odd characters. This was done to ensure that, i.e., "*fish.*" and "*Fish!*" would be recognized as identical words. I then proceeded to write a while-loop to find collocates for the specified window size and keyword. After this, I then progressed to do a series of for-loops with various steps of data-wrangling and mathematical calculations to compute collocate-metrics for all words within the specified window-size of the keyword. The most notable of these scores is the Mutual Information (MI) score. The MI-score is a metric devised to quantify the association between a given keyword and a collocate. This is done through a calculation, combining how often the two words appear together and how often they appear separately.

Using *pandas*, the script lastly writes a csv-file containing the collocate-word, the raw frequency (i.e. O11-score) and the Mutual Information score. This csv-file is written to a folder called "out" which is also created by the script. To solve the bonus assignment, I used argparse to enable the user to specify arguments from the terminal. With argparse, I made it possible for the user to specify their own *custom data-filepath*, *keyword* and *window-size* with the arguments *--filepath*, *--keyword* and *--window_size*, respectively.
For testing the keyword-argument, I recommend using less common keywords to reduce the runtime of the script. The default keyword is *"california"*.

1.4 Usage

1.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section in this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 2:

```
cd {root directory (i.e. cds-language-exam-2021)}
cd assignment_2
python3 collocation.py
```

You can specify the following optional arguments from the terminal:
Custom filepath:

```
"-f", "--filepath"
default = "data"
type = str
help = string, path to text-corpus. Texts need to be .txt-files. Be wary of difference
in operating systems in terms of specifying path with "/" or "\".
```

Keyword:

```
"-k", "--keyword"
default = "california"
type = str
help = string, the keyword for which you wish to find collocates and calculate metrics.
Needs to be lowercase.
```

Custom Window size:

```
"-w", "--window_size"
default = 2
type = int
help = integer, collocate window-size.
```

You can also type: `python3 collocation.py -h` for a detailed guide on how to specify script-parameters.

1.4.2 Repository structure and contents

The repository contains the following folder:

Folder	Description
`data/`	Folder containing a dataset consisting of 100 classic English novels in .txt-format.

Furthermore, it holds the following files:

File	Description
`collocation.py`	The python script for the assignment.
`README.md`	A README-file containing the same information as you are currently reading.

1.5 Discussion of results

I succeeded in creating a script that calculates various collocate-metrics for specific keywords and window-sizes and automatically saves selected metrics in a csv-file. Furthermore, I solved the bonus assignment by making it possible to specify script-parameters from the terminal. When running the script with default parameters (keyword = california, window-size = 2), the collocate with highest MI-score is yellowstone, which is a national park in California - indicating that script works correctly.

2 Assignment 3

2.1 Github link

My code for assignment 3 can be found here: https://github.com/JohanHorsmans/cds-language-exam-2021/tree/main/assignment_3

2.2 Official description from instructor

ASSIGNMENT 3: Dictionary-based sentiment analysis with Python

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:

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

2.3 Methods

The problem of this assignment relates to mangling and processing date-time-data and calculating rolling sentiment scores for a large text corpus. I used pandas to convert the data to date-time format and arrange the headlines in chronological order. To reduce run-time, I carried out the analysis on a subset of 100.000 randomly sampled headlines. I used SpaCy text blob to calculate the sentiment-score for the headlines. After this, I batched the data together with a batch-size of 500 to make the analysis run faster. I then calculated mean sentiment score for each week and month and plotted the scores in two separate graphs (see *discussion of results*). Lastly, I also made the script write a csv-file named "*sentiment.csv*" with the score for each individual headline.

2.4 Usage

2.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section in this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 3:

```
cd {root directory (i.e. cds-language-exam-2021)}
cd assignment_3
python3 sentiment.py
```

2.4.2 Repository structure and contents

This repository contains the following folder:

Folder	Description
`data/`	Folder containing a dataset containing over a million headlines taken from the Australian news source ABC (Start Date: **2003-02-19** ; End Date: **2020-12-31**).

Furthermore, it holds the following files:

File	Description
`sentiment.py`	The python script for the assignment.
`README.md`	A README-file containing the same information as you are currently reading.

2.5 Discussion of results

Two plots are produced by the script. One for the weekly rolling sentiment score (*figure 1*) and one for the monthly rolling sentiment score (*figure 2*). The two plots show that the news data is generally slightly positive (sentiment score > 0) in spite of the fluctuations on a weekly/monthly basis. We see that the variance in sentiment score is larger on a weekly basis than on a monthly basis, indicating that the variance seen from week to week is quite similar in both directions and thus, to a certain extent, cancel each other out when analysed on a monthly basis. I have limited the y-axis of the plot to go from -0.2 to 0.2 (the full scale goes from -1 to 1) which underlines that the fluctuations seen in the plots are very small. I argue that it is a good indicator of journalistic objectivity that the sentiment score of the headlines are very close to zero with small fluctuations appearing to be mostly even in both directions when viewed on a monthly scale.

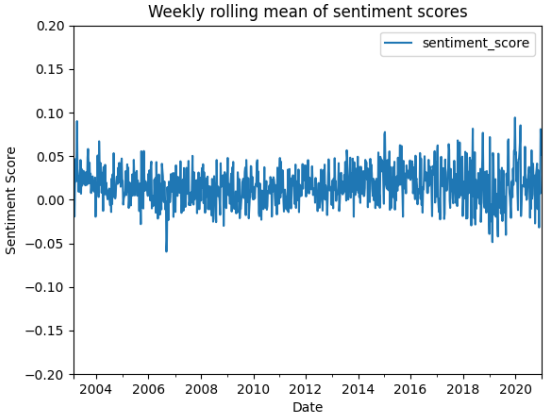

Figure 1: Weekly sentiment score.

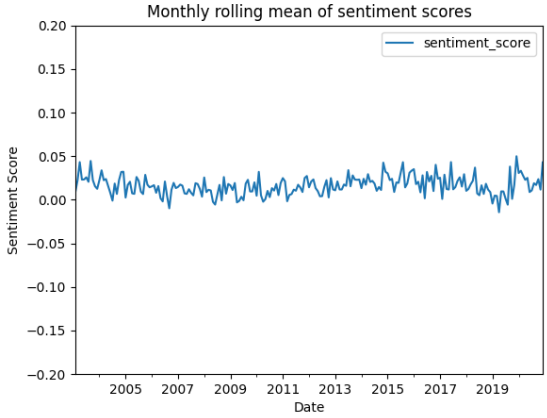

Figure 2: Monthly sentiment score.

3 Assignment 5

3.1 Github link

My code for assignment 5 can be found here: https://github.com/JohanHorsmans/cds-language-exam-2021/tree/main/assignment_5

3.2 Official description from instructor

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.

NOTE: I HAVE CHOSEN THE FOLLOWING TASK:

- 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.

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

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:

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

3.3 Contribution

I have carried out the following assignment with Gustav Aarup Lauridsen. Gustav and I contributed equally to every stage of this project from initial conception and implementation, through the production of the final output and structuring of the repository. (50/50%).

3.4 Methods

We have chosen the OffensEval2020 dataset containing 3000+ Danish comments from Ekstra Bladet and Reddit, labeled with a binary coding scheme indicating offensiveness (link: https://figshare.com/articles/dataset/Danish_Hate_Speech_Abusive_Language_data/12220805).

OffensEval2020 was a competition where researchers and data scientists from all over the world competed to create the best classification models for various languages (including Danish).

The best team in the Danish task achieved a macro F1-score of 0.8119 and the worst team achieved a score of 0.4913. For the full paper, see: https://arxiv.org/pdf/2006.07235.pdf

We wanted to create a text classifier that could classify offensive comments in Danish and compare our macro F1-score with the results from the OffensEval2020 competition. We found this task interesting due to the amount of media coverage on Danish hate speech on social media in recent months. We believe that a robust hate-speech classifier could be a very valuable tool for moderating the tone and retoric of the public debate to make it more constructive.

We trained the following models: Logistic Regression, Support Vector Machine, Neural Network, Random Forest Decision Tree (see code for further specifications). We then combined them in an ensemble where a comment was classified as offensive if the accumulated probability of all the model classifications exceeded 1.4. We have chosen to use the macro F1-score as our evaluation metric:

The F1-score is a metric devised to fuse the relation between model precision and recall into a unified score. The metric is defined as taking the harmonic mean of precision and recall. The reason for using the harmonic mean, rather than the arithmetic mean, is that the harmonic mean of a recall-score of 0 and a precision-score of 100 would result in an F1-score of 0, rather than 50. This is advantageous, since it means that a model cannot achieve a high F1-score by having a high recall or precision by itself. The macro-averaging procedure of the macro F1-score involves calculating the arithmetic mean of the F1-score for each class.

To ensure reproducible results, we used random.seed.

3.5 Usage

3.5.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section in this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 5:

```
cd {root directory (i.e. cds-language-exam-2021)}
cd assignment_5
python3 HateClass.py
```

3.5.2 Repository structure and contents

The repository contains the following folder:

File	Description
`data/`	Folder containing a testing- and training dataset consisting over a 3.000 social media comments labeled after offensiveness (i.e. *NOT* and *OFF*).

This repository contains the following files:

File	Description
`HateClass.py`	The python script for the assignment.
`README.md`	A README-file containing the same information as you are currently reading.

3.6 Discussion of results

Our ensemble containing all models achieved a macro F1-score of 0.71. It is important to note that the dataset is heavily skewed towards non-offensive comments. The skewed data is also reflected in our model predictions, where the F1-score was much higher for non-offensive comments compared to offensive ones (0.95 vs. 0.46). We believe that this bias towards non-offensive comments, might very well reflect the imbalanced nature of the dataset. It defeats the purpose of a hate-speech classifier if it categorizes such a large proportion of hate as non-offensive and, as such, we argue that there is still a lot of work to be done in spite of the seemingly honorable F1-score of 0.71.

Nonetheless it would have ranked as the 23rd best model (out of 38) in the OffensEval2020 competition, so we deem it to be quite successful when taking the circumstances into account. For an even better model, see our self-assigned project, where we achieve a macro F1-score of 0.78 on the same dataset.

4 Assignment 6

4.1 Github link

My code for assignment 6 can be found here: https://github.com/JohanHorsmans/cds-language-exam-2021/tree/main/assignment_6

4.2 Official description from instructor

Text classification using Deep Learning

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: https://www.kaggle.com/albenft/game-of-thrones-script-all-seasons

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.

Tips

- Think carefully about the kind of preprocessing steps your text data may require and document these decisions.
- Think just as carefully about the kind of parameters you use in you model. They all make a difference!
- Some sentences are very short; some are longer. Think about how you should handle this.

General instructions

- You should upload standalone .py script(s) which can be executed from the command line - one for the LogisticRegression model and one for the DL model.
- 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:

- how to build CNN models for text classification;
- how to use pre-trained word embeddings for downstream tasks;
- how to work with real-world, complex cultural text data.

4.3 Methods

The problem of the assignment relates to classifying complex textual data using, respectively, a logistic regression- and a deep learning classifier. For both models, I preprocess the data and split it into a training- and testing set (with 25% of the sentences allocated to the testing-set). For the logistic regression classifier, I use the LogisticRegression-model from the sklearn.linear_model module. For the deep learning classifier, I utilize the pretrained *GloVe-word embeddings*.

Word embeddings are basically a big vector representation of how words are located in an abstract word-space, where related entities are placed in close proximity of each other. Such a vector representation implies (conceptually) that if one, for example, took the embedding for the word king and subtracted the embedding for the word man and followingly added the embedding for woman, one should be left with the embedding for the word queen, i.e.:

$$king - man + woman = queen$$

The reason for using these embeddings is that this representation of how words are related to one another should help my model make better predictions. By computing how different words relate to each other, the model can potentially infer how classifications of some sentences can extend to other similar sentences. This is a phenomenon referred to as transfer learning. To utilize the pre-trained embeddings on a classification task, such as this, one just needs to add a neural network to the end of the GloVe embedding layer.

The neural network that I added consisted of the following; a convolutional layer with ReLU-activation, a max-pooling layer, a dense layer with 128 neurons and ReLU-activation and an output layer with 8 nodes (1 for each class).

For the Logistic Regression model i vectorize the data and transfrom it to lowercased unigrams and bigrams with tf-idf transformation. For the Deep Learning model i binarize the labels and tokenize the text. I then convert the data to an embedding matrix with either 50, 100, 200 or 300 dimensions (specified as an argument from the terminal).

For the deep learning model, I made it possible to specify training epochs and embedding-dimensions from the terminal using argparse.

I have decided to use the macro F1-score as my evaluation metric. For the reason behind this, please refer to *"assignment 5, methods"*

4.4 Usage

4.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section in this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 6:

```
cd {root directory (i.e. cds-language-exam-2021)}
cd assignment_6
python3 GoT_lr.py
python3 GoT_dl.py
```

You can specify the following optional arguments from the terminal in the `GoT_dl.py` -script:
Epochs:

```
"-e", "--epochs"
default = 10
type = int,
help = "int, number of training epochs for the neural network [DEFAULT]: 10"
```

Embedding-size:

```
"-es", "--embedding_size",
default = 50,
type = int,
```

```
help = "int, the size of the word embeddings loaded from the the GloVe-model.
Options: 50, 100, 200, 300 [DEFAULT]: 50")
```

4.4.2 Repository structure and contents

The repository contains the following folder:

File	Description
`data/`	Folder containing a csv-file with the script from every season of *Game of Thrones*.

This repository contains the following files:

File	Description
`GoT_lr.py`	The python script for the logistic regression model in this assignment.
`GoT_dl.py`	The python script for the deep learning model in this assignment.
`README.md`	A README-file containing the same information as you are currently reading.

4.5 Discussion of results

For the logistic regression model, I achieved a macro F1-score of 0.25. For the deep-learning model, I achieved a macro F1-score of 0.20 after training for 20 epochs using a word embedding size of 300 features. These results are not impressive but they act as a nice illustration of the complexity of the task. To improve results, I hypothesize that it would be beneficial to use word enitity recognition to extract the names present in the sentences and use that as a predictor of the season. Especially for a show such as Game of Thrones where many characters die in every season, I believe that this could greatly improve results.

5 Self assigned

5.1 Github link

My code for my self assigned project can be found here: https://github.com/JohanHorsmans/cds-language-exam-2021/tree/main/self_assigned

5.2 Contribution

I have made and designed the project assignment with Gustav Aarup Lauridsen. Gustav and I contributed equally to every stage of this project from initial conception and implementation, through the production of the final output and structuring of the repository. (50/50%)

5.3 Project description

Danish hate speech detection:

For our self-assigned project, we wish to see if we can improve the Danish hate-speech detection algorithm that we designed for assignment 5. As stated in assignment 5, we find this task very interesting due to the amount of media coverage on Danish hate speech on social media in recent months. We believe that a robust hate-speech classifier could be a very valuable tool for moderating the tone and retoric of the public debate to make it more constructive.

In assignment 5, we achieved a macro F1-score of 0.71. The current state-of-the-art, as described in OffensEval2020, achieves a macro F1-score of 0.81. Our goal with this project is to build a competing state-of-the-art model with similar performance and make it openly available by uploading it, as the first Danish hate speech detection model, to huggingface.co. We wish to do this using the Nordic BERT-architecture by BotXO.

Following this we are going to build a .py-script that can be employed for hate speech classification on ones own dataset. Furthermore, we are creating a Jupyter notebook acting as a tutorial to easily help users deploy the model from huggingface on their own data. Using our huggingface-model will be advantageous, since it takes a long time to train a BERT-model for classification tasks. Using our pretrained model from huggingface, will make it easier and much less time-consuming to implement hate speech moderation for various media-sites and firms who wish to combat Danish hate speech on their online platforms. To improve usability, we will make the model compatible with both a tensorflow- and pytorch framework

In summary, the project is comprised of the following steps:

1. Train and test a Nordic Bert-model on the official OffensEval2020-dataset
2. Upload the trained model to huggingface.co
3. Create a Jupyter notebook and .py-script designed to help users deploy the model on their own data.

5.4 Methods

NOTE: Some parts of the following section is repeated from assignment 5
For model training and testing, we are using the OffensEval2020 dataset containing 3000+ Danish comments from Ekstra Bladet and Reddit, labeled with a binary coding scheme indicating offensiveness (link: https://figshare.com/articles/dataset/Danish_Hate_Speech_Abusive_Language_data/12220805).

OffensEval2020 was a competition where researchers and data scientists from all over the world competed to create the best classification models for various languages (including Danish).

The best team in the Danish task achieved a macro F1-score of 0.8119 and the worst team achieved a score of 0.4913. For the full paper, see: OffensEval2020.

To make our model-performance comparable to the current state-of-the-art presented in OffensEval2020, we utilized macro F1-score as our evaluation metric:

The F1-score is a metric devised to fuse the relation between model precision and recall into a unified score. The metric is defined as taking the harmonic mean of precision and recall. The reason for using the harmonic mean, rather than the arithmetic mean, is that the harmonic mean of a recall-score of 0 and a precision-score of 100 would result in an F1-score of 0, rather than 50. This is advantageous, since it means that a model cannot achieve a high F1-score by having a high recall or precision by itself. The macro-averaging procedure of the macro F1-score involves calculating the arithmetic mean of the F1-score for each class.

For our modeling, we have chosen to use the Nordic BERT-architecture. The reason behind using Nordic BERT is that it has been deployed with great results in the litterature for a large range of similar classificaion

tasks. Furthermore, the winning team in the OffensEval competition for the Danish task also used a Nordic BERT framework.

We trained the the BERT model for 10-epochs with the following hyperparameters:

- Learning rate: 1e-5,
- Batch size: 16
- Max sequence length: 128

We ran- and developed the code on Google Colaboratory. For our model-training notebook, please see: "dk_hate_training.ipynb"

Our uploaded model can be found here, on huggingface.co: https://huggingface.co/Guscode/DKbert-hatespeech-detection

5.5 Usage

5.5.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section in this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

To evaluate the model, please refer to the *dk_hate_detect.py*-script, since this is the main tool. The *dk_hate_detect.ipynb*-notebook is mainly designed as a tutorial for non-expert users. Both contain the same model.

To run the script go through the following steps (**NOTE:** you have to specify either the *--text*-argument or the *--data* and *--column*-arguments to run the script):

```
cd {root directory (i.e. cds-language-exam-2021)}
cd self_assigned
python3 dk_hate_detect.py
```

You can specify the following arguments from the terminal in the `dk_hate_detect.py` -script:
Data path:

```
"--column"
required = False
default = None,
help = "name of column including text for hatespeech detection "
```

Column:

```
"--column"
required = False
```

```
default = None,
help = "name of column including text for hatespeech detection "
```

Single string classification:

```
"--text"
required = False,
default = None
type = str
help = "string for single string hatespeech detection"
```

Output:

```
"--output"
required = False
type = str, default = "./"
help = "output path for dataset with hatespeech column"
```

You can also type: `python3 dk_hate_detect.py -h` for a detailed guide on how to specify script-parameters.

Go through the following steps to run the notebook:

1. Navigate to the "self_assigned"-folder.
2. Open the "dk_hate_detect.ipynb"-file.
3. Make sure the kernel is set to "lang_venv".
   - You can do this by pressing "kernel" → "change kernel" → "lang_venv".

5.5.2 Repository structure and contents

The repository contains the following folder:

File	Description
`data/`	Folder containing a testing- and training dataset consisting over a 3.000 social media comments labeled after offensiveness (i.e. NOT and OFF).

Furthermore, it holds the following files:

File	Description
`dk_hate_detect.py`	The python script for the assignment.
`dk_hate_detect.ipynb`	The Jupyter notebook for the assignment.
`dk_hate_training.ipynb`	The Jupyter notebook we created when training the model.
`README.md`	A README-file containing the same information as you are currently reading

5.6 Discussion of results

Our model achieved a macro F1-score of 0.78. As stated in assignment 5, it is important to note that the dataset is heavily skewed towards non-offensive comments. This skew is also reflected in our model predictions, where the F1-score was much higher for non-offensive comments compared to offensive ones (0.95 vs. 0.60). We believe that this bias towards non-offensive comments, might very well reflect the imbalanced nature of the dataset.

As stated earlier, the currently best performing Danish hate speech model achieved a macro F1-score of 0.81 on the same dataset (as described in OffensEval2020). As such, we have not quite built the new gold-standard model for hate speech detection in Danish. Nonetheless, we have come very close, and our model would have finished 4th place (out of 38 contenders) in the OffensEval2020 competition. Furthermore, it is important to note that we have created the best **publically** available Danish hate speech + a ready-to-use .py-script and a thorough Jupyter notebook tutorial on how to use it. Therefore, we argue that we have greatly improved the possibilities for an actual real-life implementation of such an algorithm.