Maastricht University

Information Retrieval and Text Mining

Semester 2

Text Mining the Bible

Project Report

Group
Francesca Battipaglia 6263045
Christos Kaparakis 6258164

July 1, 2021

Contents

1 Introduction

The Bible, also known as the Holy Bible, is a group of religious texts of Judaism and Christianity, it contains both the Old Testament and the New Testament. It is the best-selling book of all time, having sold around 5 billion copies to date.

The goal of this project is to analyze the text of the Bible's books and answer some questions that arise for two topics.

- How does the text evolve through the years?
- Which are the differences between the Old and New Testament?

The above-mentioned questions led the whole development of the project. We used different techniques to try to answer these questions, and the most interesting ones turned out to be Sentiment Analysis and Named Entity Recognition.

2 Preprocessing

The Bible version used in this project is the King James Version (KJV). The first step we took was to acquire the text in a data frame[1] consisting of every verse together with useful information such as in which book and chapter it belongs to.

id	b	c	v	t
1001001	1	1	1	In the beginning God created the heaven and the earth.

Figure 1: Example of the data set

In the second step, two new columns were created. One that included the text without punctuation for every verse and another containing the count of words found in a verse.

Figure 2 exhibits the information we are able to extract following this preprocessing.

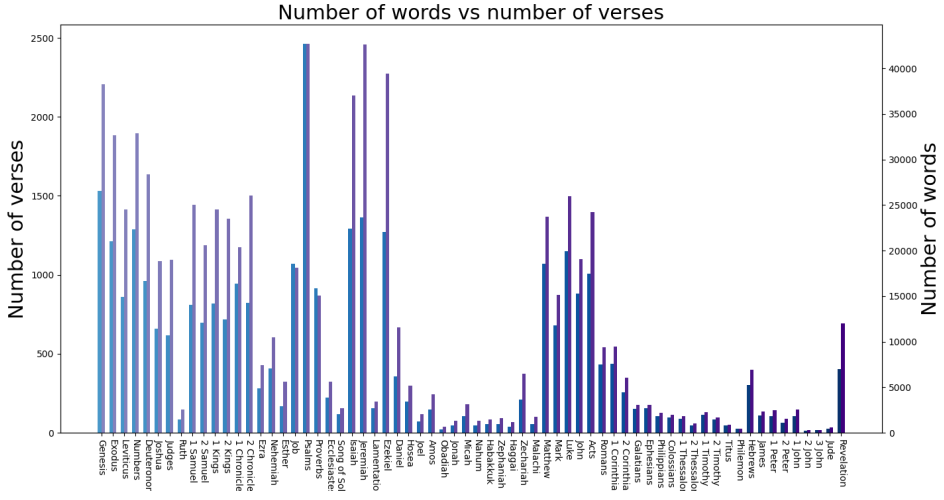


Figure 2: Barplot showing the number of words and the number of verses for every book of the Bible. The blue bars display the number of verses in a book, while the purple bars the number of words contained in a book.

In the third step of the preprocessing, we took advantage of the SpaCy package. SpaCy (Honnibal, Montani, Van Landeghem, & Boyd, 2020) is an open-source software library that

[1] https://www.kaggle.com/oswinrh/bible

is developed for performing simple to advanced Natural Language Processing (NLP) tasks, such as:

- transforming all the text to lowercase;
- tokenization;
- lemmatization;
- POS tagging.

These techniques provided us with the means to create visualizations, such as the Figures 10, 11, 12, but also more advanced visualization that were used for the creation of our project's video. Some captures of these graphs can be found in the Appendix (Section 9).

Below there is a more detailed explanation of the aforementioned processes which we implemented in this project.

Tokenization is loosely referred as segmenting a text document into words, that are also known as tokens. A token is defined as a sequence of characters that together forms a semantic unit while processing of text. In some contexts tokens may not be words and might comprise of sub-words or even multi-word phrases depending on what is considered as a semantic unit(Webster & Kit, 1992). SpaCy tokenizes a given text into words and punctuations by applying rules that are tuned for a specific language. The most basic rule for English language is to split a sequence of characters in text at white spaces (e.g. blank-space), often known as white space tokenizer. SpaCy exactly does that to start the tokenization process and then builds on top of it by iterating over the sub-strings from left to right and looking for rules for splitting.

Lemmatization in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. For example, *run, runs, ran and running* are forms of the same set of words that are related through inflection, with *run* as the lemma.

Part-of-speech (POS) tagging is the process of tagging a word with its corresponding part-of-speech like noun, adjective, verb, adverb, etc., following the language's grammatical rules that are further constructed on the basis of the context of occurrence of a word and its relationships with other words in a sentence.

After tokenization SpaCy can provide the lemma, but also tag a given Doc object using its state-of-the-art statistical models. The lemmatized words and the tags are available as attributes of a Token object.

3 Classification

For this project we decided to implement two different classification tasks, using the state-of-the-art model, BERT. This section will elaborate on what we classified and how we did it.

3.1 Classification of Old and New Testament

The Christian Bible contains 66 books and is divided into two parts:

- before Jesus: *Old Testament* with the history of Israel and its people;
- after Jesus: *New Testament* with stories about Jesus (gospels) and some others about disciples.

Since each verse is identified by the number of the verse, the chapter, and the book, it is possible to check what are the verses that belong to the New and Old Testament. For this reason, our data set has been enriched with a column that identifies the Testament, for each verse. This can be done because we know that the books $1-39$ are from the Old Testament and books $40-66$ are from the New Testament.

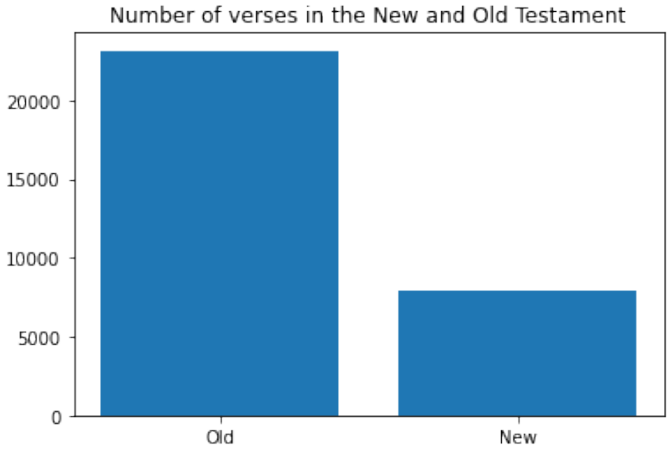


Figure 3: Number of verses per testament.

In Figure 3, we can see how the number of verses are distributed between the New and the Old Testament.
The classification task that we want to achieve is to predict whether a verse belongs to the New or the Old Testament.

For this classification task we used the Bidirectional Encoder Representations from Transformers (BERT)(Devlin, Chang, Lee, & Toutanova, 2018), released by Google AI Language which represents the state-of-the-art in several fields of Natural Language Processing.

The reason why we decided to go for BERT, even though we do not have a lot of data (31103 verses), is because of the length of our documents. In our case, documents are represented by verses, which have on average a length of 25 words. BERT is said to have good performance on short text by (Ye, Jiang, Liu, Li, & Yuan, 2020). Indeed, for short text and classification tasks, deep learning is said to outperform statistical and geometric models. Taking all of this into account, we decided to experiment the state-of-the-art model and its performance on short text.

In order to use BERT, we had to reshape our inputs in order to be considered as valid inputs for the BERT model. In particular we had to create three different inputs, namely the *Token Embedding*, the *Segment Embedding* and the *Position Embedding*. The first one is a vector representation of the words, after the tokenization has been applied. The second one is

needed to help BERT to distinguish between the actual words and the padding. Finally, the Position Embedding is used to let the model know about positions of the words in a sentence.

The second dimension of these inputs is the same, and we choose the value 186, since it represents the maximum number of tokens in the first input, multiplied by 1.5. This led two three different inputs, which are then passed into the Keras layer. We used softmax as activation function of our model, such that the same model could also be used for the other classification task 3.2.

An idea of what the structure of our model is can be seen in the below picture[2] 4.

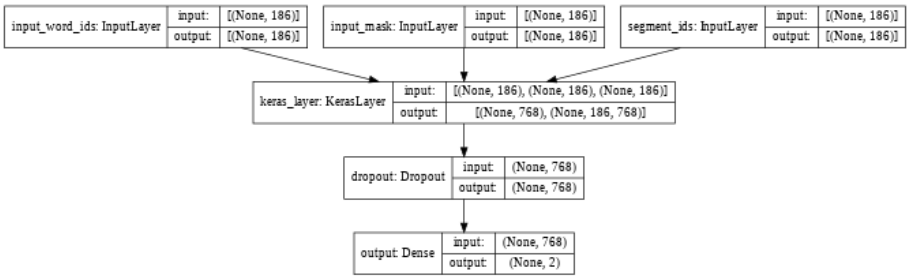


Figure 4: Model architecture.

The model has been trained on 80% of the data, and the remaining part was saved as test set. The number of epochs chosen is 3 because we wanted the model to properly adjust the weights. However, we noticed an increase of the validation loss between the second and third epoch and, since we did not want the model to over-train, we reduced the number of epochs for the training to 2.

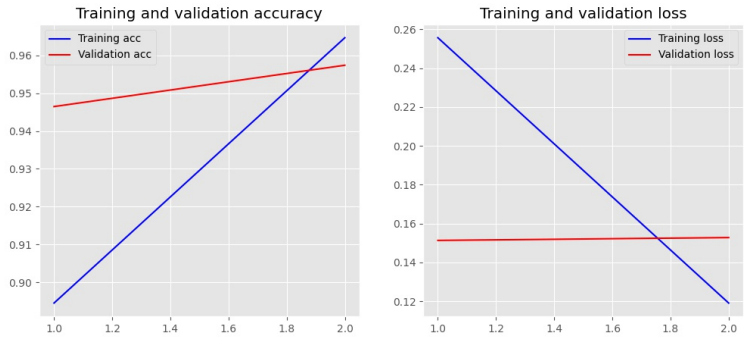


Figure 5: Accuracy and loss plotted against epochs.

As can be seen in the above plot (Figure 5), throughout the epochs, the validation accuracy grows. Moreover the other metrics computed during the training, can be seen in the below code.

```
Epoc 1/2
    loss: 0.2556 - accuracy: 0.8945 - auc: 0.9620 - precision: 0.8945 - recall
    : 0.8945

Epoch 2/2
    loss: 0.1190 - accuracy: 0.9647 - auc: 0.9908 - precision: 0.9647 - recall
    : 0.9647
```

The results obtained were as good also in the test process, this proved that the number of epochs chosen was correct, because the performance of the model is still good also on unseen instances.

[2]However this picture does not include the inner structure of BERT.

```
'accuracy': 0.9840044975280762, 'auc': 0.9969353079795837, 'precision':
    0.9840044975280762, 'recall': 0.9840044975280762
```

To conclude, even if it is not possible to interpret why BERT produces certain results, the main advantage of this model is, with no doubts, the high accuracy that it leads.

3.2 Classification of the books

For this classification task, the same approach as the one used for the New/Old Testament classification has been used. The structure of BERT did not change, apart from the number of classes, which were considerably higher, going from 2 to 66 as we can see in Figure 6.

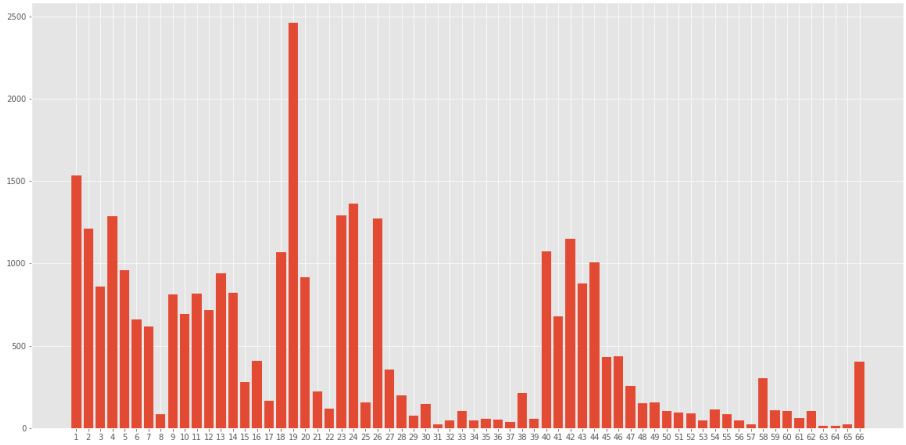


Figure 6: Bar chart of number of verses per book.

Obviously this task was more computationally intensive, given the increase in the number of parameters the model was trained on. Another issue with this model could have been the high class imbalance, caused by the multitude of the labels.
However, we decided to train this model anyway, to test the performance of BERT and compare it to the binary classification task.

In the code below, the metrics computed after each epoch can be seen. We can say that BERT scores pretty well also for this task, since both the precision and the recall increase overtime. We first trained the model on the same number of epochs of the previous task, so 3, but we noticed that the model was not performing good, because the number of parameters were considerably higher. Hence, we then trained on 5 epochs, since we wanted to improve the performance, but also to not let the model to over-train.

After 5 epochs, the metrics that we got after each of the epochs can be seen in the code below.

```
Epoc 1/5
    loss: 2.8542 - accuracy: 0.2623 - auc: 0.8839 - precision: 0.7028 - recall
    : 0.0738
Epoch 2/5
    loss: 1.7646 - accuracy: 0.4914 - auc: 0.9612 - precision: 0.7513 - recall
    : 0.2941
Epoch 3/5
    loss: 1.3483 - accuracy: 0.5942 - auc: 0.9775 - precision: 0.7953 - recall
    : 0.4305
Epoch 4/5
    loss: 1.0654 - accuracy: 0.6772 - auc: 0.9861 - precision: 0.8349 - recall
    : 0.5323
Epoch 5/5
    loss: 0.8734 - accuracy: 0.7368 - auc: 0.9913 - precision: 0.8740 - recall
    : 0.6084
```

We also collected the evolution of these metrics throughout the epochs, and this can be seen in Figure 7.

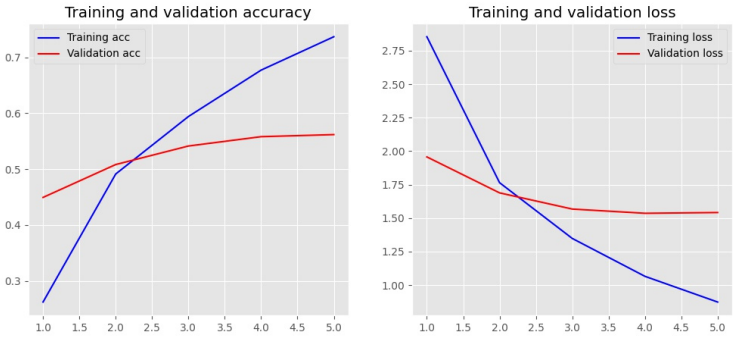


Figure 7: Accuracy and loss plotted against epochs.

More interesting metrics to look at are the ones obtained during the testing. BERT proved to score well also on unseen entries, with a high precision and recall. However, the most interesting one is the "auc", the area under the ROC, which is close to a perfect classifier.

```
1 'accuracy': 0.8093400597572327, 'auc': 0.9949079155921936, 'precision':
    0.9120547771453857, 'recall': 0.7173056602478027
```

To conclude, we can say that the performance of BERT are good in both classification tasks. The model showed that it can be successfully used to classify short text, even without the amount of data that usually is needed to obtain high performance. Moreover, the performance of the model are still good even with a high amount of labels and only five epochs of training.

4 Sentiment Analysis

Sentiment analysis is the task of determining the emotional value of a given expression in natural language. It is essentially a multi-class text classification where the given input text is classified into a *positive*, *neutral*, or *negative* sentiment.

In this project, we aim to analyse the sentiment by using two open-source libraries, namely TextBlob (Loria, 2020) and VADER (Hutto & Gilbert, 2014) and draw comparisons between them.

4.1 spaCyTextBlob

spaCyTextBlob is a pipeline component that enables sentiment analysis using the TextBlob library. TextBlob is a simple python library that offers API access to different NLP tasks such as sentiment analysis, spelling correction, etc.

TextBlob sentiment analyzer returns two properties for a given input sentence:

- **Polarity** is a float that lies between $[-1, 1]$, -1 indicates negative sentiment and $+1$ indicates positive sentiments.
- **Subjectivity** is also a float that lies in the range of $[0, 1]$. Subjective sentences generally refer to personal opinion, emotion, or judgment.

By using the integrated pipeline in spacy we can easily extract the polarity and subjectivity for every token processed in every verse. We are only going to focus on polarity.

Following the above, we extracted the polarity for every word in every verse and averaged that to get the verse's overall sentiment score.

4.2 VADER

Valence aware dictionary for sentiment reasoning (VADER) is another popular rule-based sentiment analyzer. It uses a list of lexical features (e.g. words) which are labelled as positive or negative according to their semantic orientation to calculate the text sentiment. Vader sentiment returns the probability of a given input sentence to be *Positive*, *Negative*, and *Neutral*.

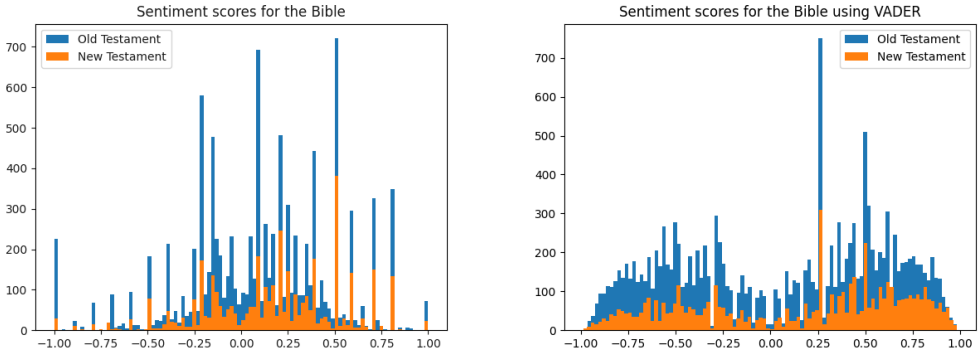
For example:
"But his flesh upon him shall have pain and his soul within him shall mourn"
Positive : 0%
Negative : 38.5%
Neutral : 61.5%

To compare with the previously used package TextBlob, the polarity of the preceding sentence was 0.

The reason why we chose to complement our sentiment analysis with a secondary method, is because we recognised that in a lot of instances, where "negative" words were used, the TextBlob package was returning a polarity of 0, meaning that it wasn't classifying them correctly.

The graph below shows the distribution of the sentiment scores for the Old and New Testaments using the aforementioned methods.

As we can see from the two figures in Figure 8, by using VADER's advanced lexicon we are able to classify more words in the text as to their sentimentality, rather than just given them a score of 0 like the TextBlob package does. For this reason, we will further use the extracted data only from the VADER sentiment analyzer.



(a) TextBlob (b) VADER

Figure 8: Frequency plot of the sentiment scores for the Old and New Testament

5 Named Entity Recognition

Named Entity Recognition (NER) deals with identifying in the text *entities*, where an entity can be described as a class to which different words belong to. Example of entities are: *Person, Organisation and Location.* This section describes the different approaches we tried to get entities extracted from the text.

5.1 NER with Spacy

Among the available open-source libraries, we decided to start with SpaCy[3]. However, even if the implementation of NER with Spacy was very easy to implement, it was not performing as expected (see Table 1). This seemed very strange, since SpaCy is a well-known and good performing library used in different fields.

For this reason, we researched and found that, for some libraries, it is actually difficult to extract biblical names. Obviously, the corpora, on which these libraries have been trained on, are modern language ones.

As explained in (Mendels, May 8 2020), the reason why SpaCy and other libraries do not perform well on biblical texts is that

1. "biblical names are sometimes older and less common (therefore might be less frequent in the data set the model was trained on)."
2. "the surrounding sentence is less likely to co-occur with the specific name on the original data set."
3. "issue with the data set itself (such as wrong annotations by humans)."

5.2 NER with FLAIR

One way to solve the issue of the poor performance of some NER models is to augment the data these models have been trained on, adding, for example, also names that are not popular in our modern language. Indeed, that what has been done in (Mendels, May 8 2020), where they tested that is actually the data set a model has been trained on that makes a difference. Here, they show that some models perform good only when they are trained on specific data sets. In particular, it was shown that the one that performed the best is a Flair CONLL-03[4] based model.

[3] `https://spacy.io/usage/spacy-101#pipelines`
[4] `https://www.clips.uantwerpen.be/conll2003/ner/`

For this reason, we decide to check if the Flair model was actually performing good in extracting entities for our project. **Flair**(Akbik et al., 2019) is a simple natural language processing (NLP) library developed and open-sourced by Zalando Research[5]. Flair's framework builds directly on PyTorch, one of the best deep learning frameworks out there. The Zalando Research team has also released several pre-trained models for NER.

The number of extracted entities from both frameworks can be seen in Table 1. More specifically, using the Flair library we were able to extract 233% more entities overall, and 275% more PERSON entities compared to SpaCy.

Library	PER	LOC	Total
SpaCy	8889	6150	15039
FLAIR	24474	10582	35056

Table 1: Number of extracted entities using the SpaCy and FLAIR frameworks.

This shows a very big increase in the number of extracted entities using the different methods, but, in order to give some context as to why this is important, lets see an example at how the different methods are able to identify a specific person, and maybe the most important one, *Jesus Christ*.

The number of appearances of 'Jesus', 'Christ' or 'Jesus Christ' in the extracted entities using FLAIR is 1198. On the contrary, the number of appearances of 'Jesus', 'Christ' or 'Jesus Christ' in the extracted entities using SpaCy is 0.

To conclude, this task, Named Entity Recognition, showed how the Bible could be classified as a "long tail problem", where there are many exceptions and few training samples available[6]. Indeed, the Bible is written in a language that is completely different to the modern one. Hence, there is no particular attention in training models on data set that contain this kind of vocabulary. Hence, even a very well performing library as Spacy can fail when applied to a very specific problem.

[5] `https://github.com/flairNLP/flair`
[6] where the few training samples available has to be intended as few data set containing this lexicon.

6 Validation

For our project we used the King James Version Bible version. Since we extracted information from the text, there was no test set to validate our results. For this reason, we created our own test set in order to validate the outcomes we had for the *Named Entity Recognition* and the *Sentiment Analysis*.

We decided to not include the two Classification tasks in our validation set because we had already a measure of the performance of the model, thanks to the metrics we computed: accuracy, precision and recall. Hence, regarding Classification, we based our validation on those metrics, since they are, with no doubt, more accurate than how we could have been.

As mentioned before, for the NER and the Sentiment Analysis we used our own validation data set. We selected 70 random verses from the text, we created an Excel file, and individually validated our results. In Figure 9, the first two rows of our validation set is presented.

	id	b	v	t	Testament	POS	ner	sent_categorical
15285	19086001	19	1	Bow down thine ear, O LORD, hear me: for I am poor and needy.	Old	['NOUN', 'AI []		Negative
30932	66014005	66	5	And in their mouth was found no guile: for they are without fault b	New	['CCONJ', 'A [<PER-span (19): "God">]		Positive

Figure 9: Example of the entries in the validation set.

The way we used to validate both the NER and the Sentiment Analysis, was either to *accept* or *reject*, and the convention we used was to have 0 and 1 respectively, in order to avoid typos and make our validation more robust and easier to be processed later.

In the below tables 2, 3, the results of the validation process are presented.

	Judge 1		
Judge 2	Agree	Disagree	Total
Agree	51	6	57
Disagree	2	11	13
Total	53	17	70

Table 2: Validation of Named Entity Recognition.

	Judge 1		
Judge 2	Agree	Disagree	Total
Agree	42	14	56
Disagree	3	11	14
Total	45	25	70

Table 3: Validation of Sentiment Analysis.

Following the above two tables, we calculate the Kappa Score (aka Cohen's Kappa Coefficient) using the following formula.

$$KappaScore = (Agree - ChanceAgree)/(1 - ChanceAgree)$$

Where "Agree" is the proportion of ratings that are in agreement between the two judges, and "ChanceAgree" is the expected proportion of chance agreements.
In our case this lead to:

For NER:

$$Agree = \frac{51 + 11}{70} = 0.885714$$

$$ChanceAgree = \frac{53}{70} * \frac{57}{70} = 0.62$$

$$KappaScore = \frac{0.885714 - 0.62}{1 - 0.62} = 0.70$$

For Sentiment Analysis:

$$Agree = \frac{42 + 11}{70} = 0.757143$$

$$ChanceAgree = \frac{45}{70} * \frac{56}{70} = 0.51$$

$$KappaScore = \frac{0.757143 - 0.51}{1 - 0.51} = 0.50$$

The validation for Named Entity Recognition produces a kappa score of 0.70. We can interpret that as a substantial agreement between the judges. However, the Sentiment Analysis validation does not produce as high results, showing a moderate agreement with a score of 0.5. That can be attributed to the fact that the words used in the text of the Bible are often obsolete and ambiguous sentiment-wise, resulting to less aligned views of the judges.

7 Visualization

7.1 n-grams

For the creation of the uni-grams, firstly the so called "stop words" were removed from the list of words of our text. Stop words are a set of words that are very commonly used, such as "I", "the", "is", etc. which carry very little information. Following this preprocessing step, all the words were transformed to lowercase format and then their frequency of appearance in the whole text was counted. In Figure 10 the top 25 words (or uni-grams) for the two testaments are presented.

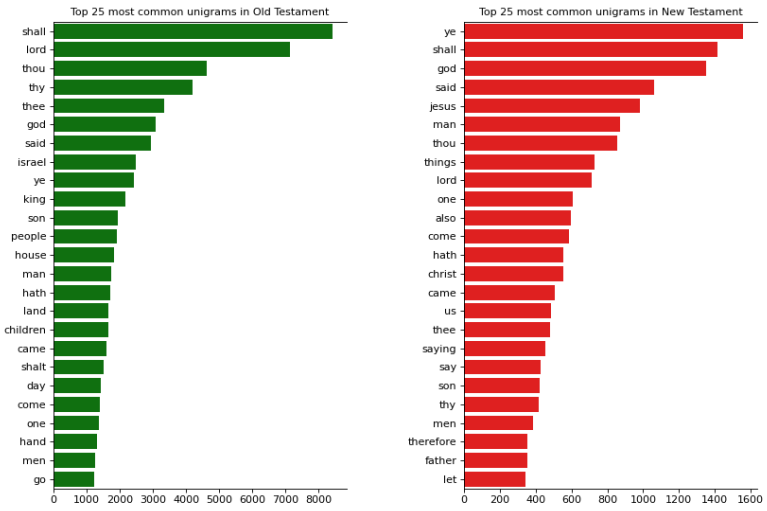


Figure 10: The 25 most frequent uni-grams of the Old and New Testaments of the Bible

Similarly to uni-grams, the bi-grams were produced by first transforming the text to lowercase format, then producing all the 2-word sequences of the text, removing the ones which contain stop-words and lastly counting their frequencies. The same procedure was followed for the tri-grams but for all the 3-word sequences of the text.

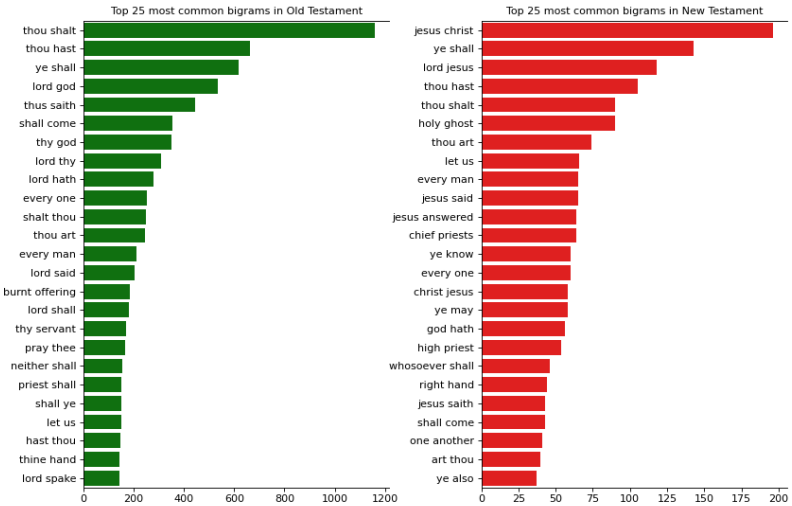


Figure 11: The 25 most frequent bi-grams of the Old and New Testaments of the Bible.

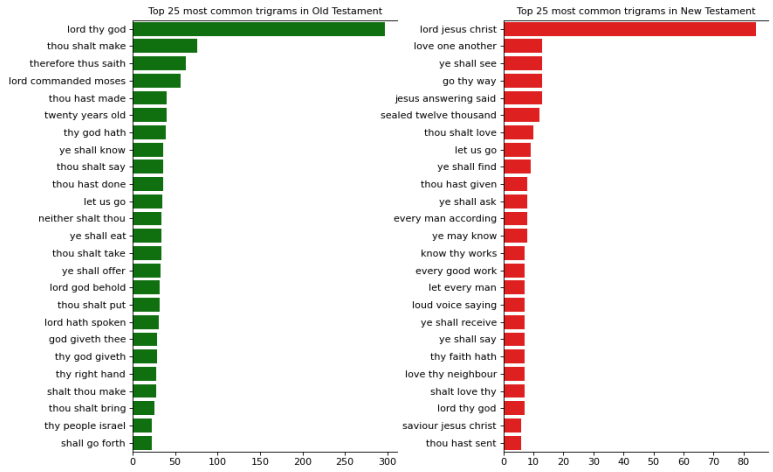


Figure 12: The 25 most frequent tri-grams of the Old and New Testaments of the Bible.

7.2 Sentiment

For the sentiment analysis, we used two different approaches, which are fully explained in the Sentiment Analysis section 4. However, for the Visualization, we decided to use the results of the best performing model, VADER.

More precisely, we are interested in showing what is the evolution of the sentiment throughout the books, both for the New and Old Testament.

Figure 13: Evolution of the sentiment throughout the Old testament.

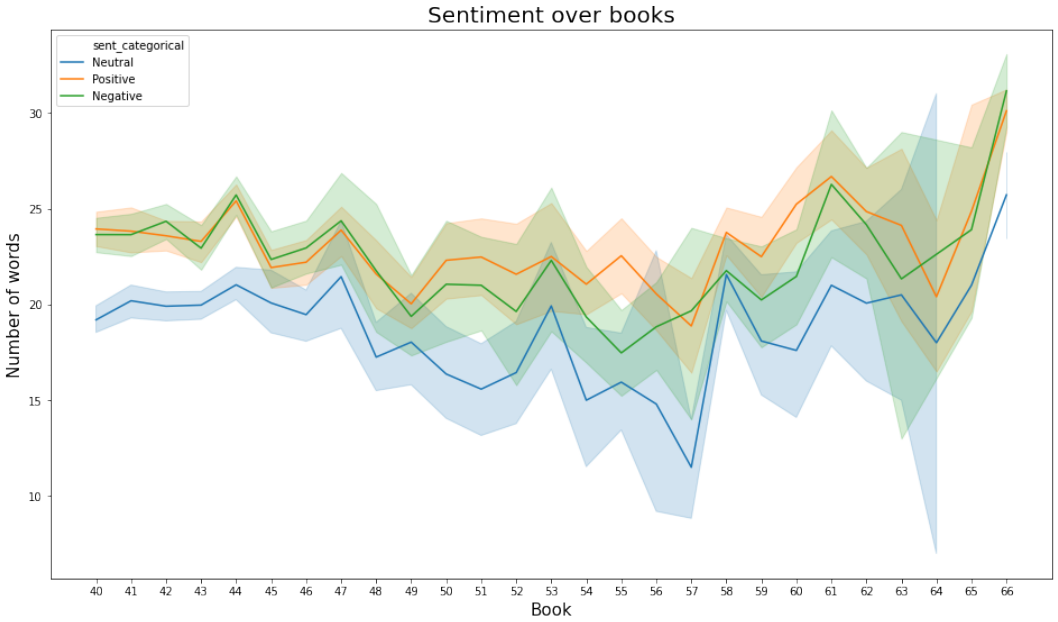


Figure 14: Evolution of the sentiment throughout the New testament.

Overall, the three classes, positive, negative and neutral are well balanced throughout the Bible. However, the validation of the Sentiment, computed through the kappa measure, shows that the Sentiment with VADER is correct in only 50% of the cases.

7.3 NER

As it was stated in previous section, the Named Entity Recognition task (with FLAIR) of the Bible resulted in 35056 extracted entities overall. In order to present our most important findings of this technique, we decided to use **stream graphs**.

Firstly, we extracted the 10 most frequent person entities and the 10 most frequent location entities, but that resulted in our graphs appearing messy. For that reason, and after experimenting with the different entities, we noticed that, using the top three entities, our stream graphs gained more reader value.

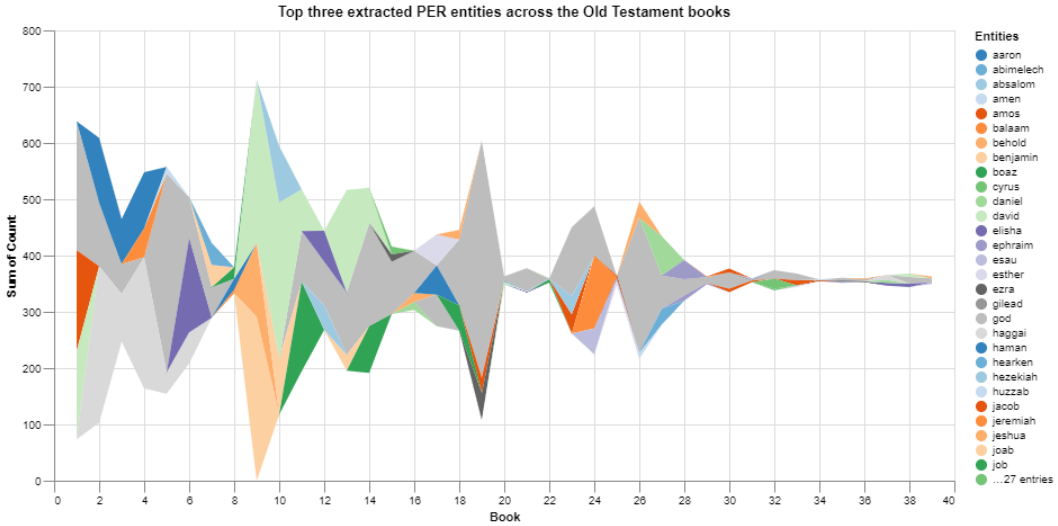


Figure 15: Stream graph of 3 most frequent person entities per book in the Old Testament.

In Figure 15 we can see that “god” is the most frequent entity for almost all of the Old Testament. Furthermore, it is clear from the stream graph that the Old Testament contains

many different stories of many different people, since the top extracted entities (besides "god") are not consistent throughout the 39 books of the Old Testament.

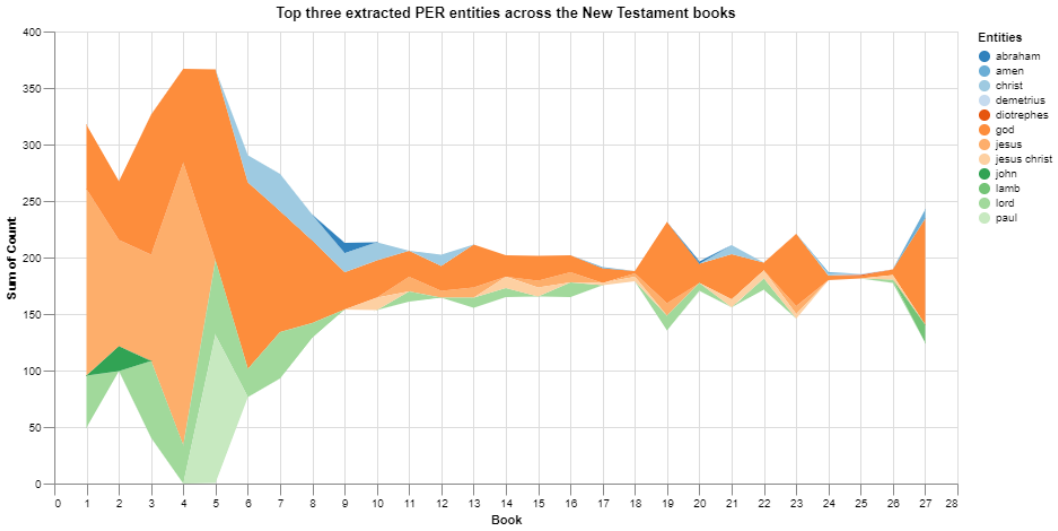


Figure 16: Stream graph of entities per book.

On the contrary, Figure 16 displays the consistency of the 27 books in the New Testament, with only 12 entities being in the 3 most frequent ones per book. Evidently, "god" and "jesus" are the most prevalent person entities throughout the New Testament.

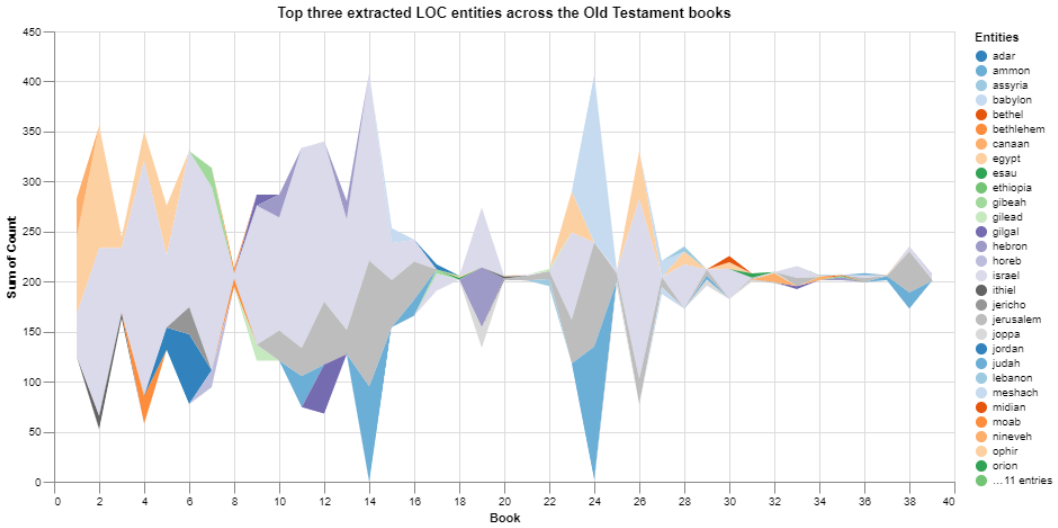


Figure 17: Stream graph of entities per book.

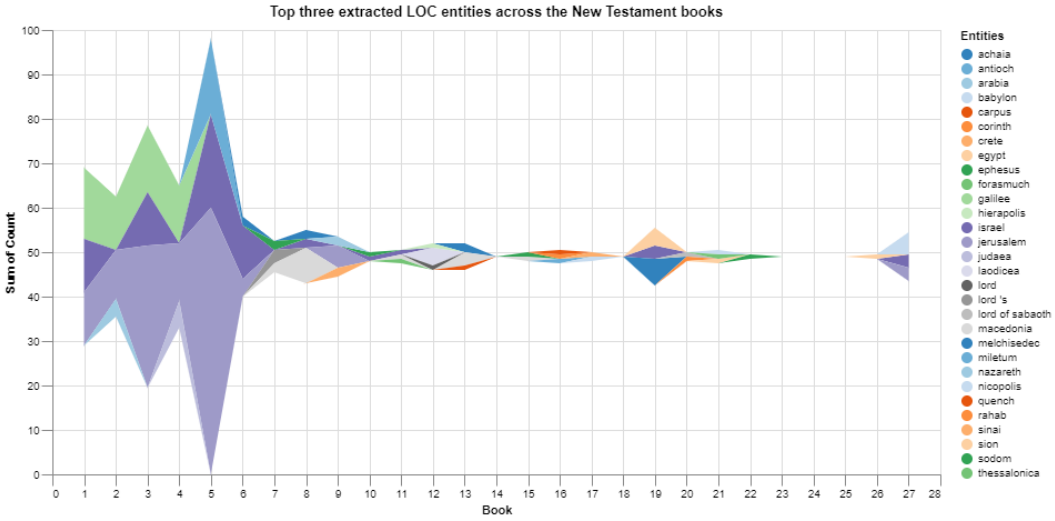


Figure 18: Stream graph of entities per book.

Figures 17 and 18 display the three most common location entities extracted from the Old and New testament respectively. It is very much clear that the two most important locations of the Bible (based on their number of occurrence in the text), is 'Israel' and 'Jerusalem'.

8 Conclusions

This section will answer the research questions formulated at the beginning of the project and will also expand on the ones that emerged throughout the course of the project.

To give a clearer context, the research questions, stated in the Introduction (Section 1), are reported also here:

- How does the text evolves through the years?
- Which are the differences between the Old and New Testament?

Regarding the first question, it was actually impossible to capture an evolution throughout the years, since the Bible does not contain specific reference to years, because it was written in the form of a story. Moreover, some books overlap because there are different writers, and they give their own point of views regarding the evolution of the events.

With respect to the second question, we were able to identify several differences between the two testaments. First of all, one big difference concerns the length of the two collections, the Old Testament, with 39 books, contains more than double the verses of the New Testament, as can be seen in plot 3.

Regarding the sentiment, we noticed that it was mostly balanced throughout the two testaments. However, this analysis may be a bit misleading, since it is very difficult to recognize the sentiment in the Bible. That can be attributed to the fact that the words used in the text of the Bible are often obsolete and ambiguous sentiment-wise, resulting in a difficulty for the models to properly recognize the sentiment.

Indeed, this was also observed in the NER analysis, where we found out that some libraries cannot properly recognize entities, because they have been trained on corpora that do not include the entities that were common in the Bible.

What we noticed in both the Sentiment analysis and NER made us reflect on the possibility to consider the Bible as a "long tail problem".

8.1 Future research

For extracting insights of the Bible through time, more research can be done on finding other data sets which contain temporal information of the Bible's books, in order to enhance the current one.

For NER, better extraction of the entities can be implemented, first be manually annotating a part of the Bible, and then training a model to extract the entities from the rest. Furthermore, it is possible to correct the extracted entities by linking the ones that refer to the same person (e.g. "Jesus" with "Jesus Christ", "God" with "Lord" etc.), but this is not something we touched upon on this project, given the time constraints.

Finally, this project aimed to provide some general insights from the whole text such as the change of sentiment throughout the book, the usage of parts-of-speech such as adjectives, nouns and verbs, building a model for classifying verses to the Books/Sentiments, and extracting entities by using state-of-the-art methods. We believe that focusing on specific characters such as "God" or "Jesus", and analyzing the text where these characters exist in, more detailed information will be able to be extracted.

9 Appendix

9.1 Project Repository

`https://github.com/ckaparakis/IRTM_Bible`

9.2 Project workload distribution

The project tasks have been equally split between the two members. Since we were both in Maastricht, we used this as an opportunity to finally work in person on a project. That's why we were able to split all the tasks equally.

9.3 Extra plots

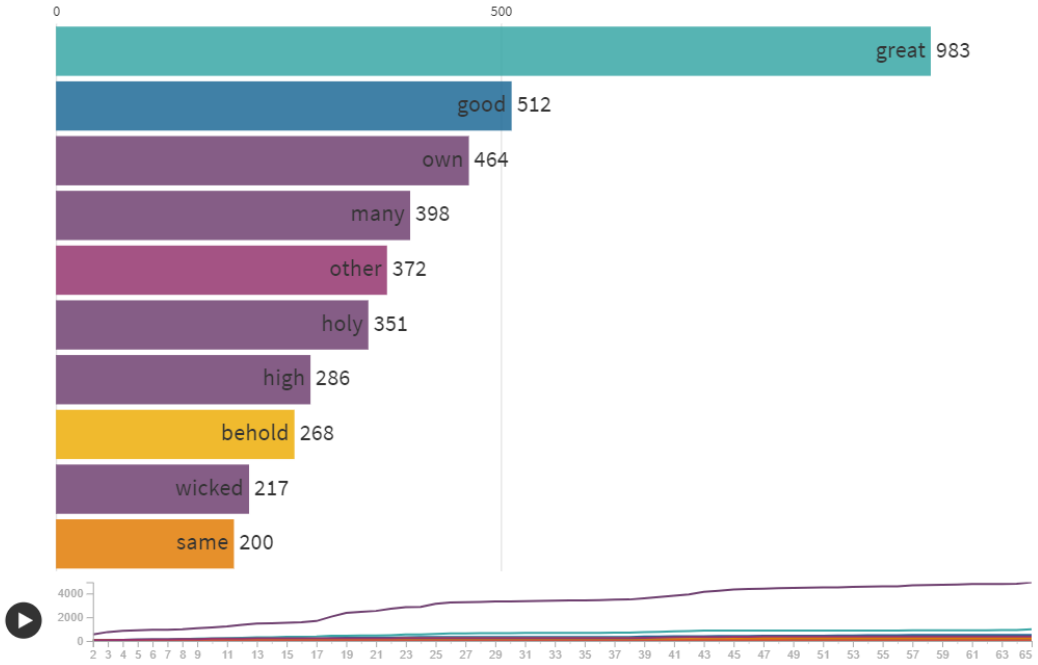


Figure 19: Most popular adjectives in the Bible

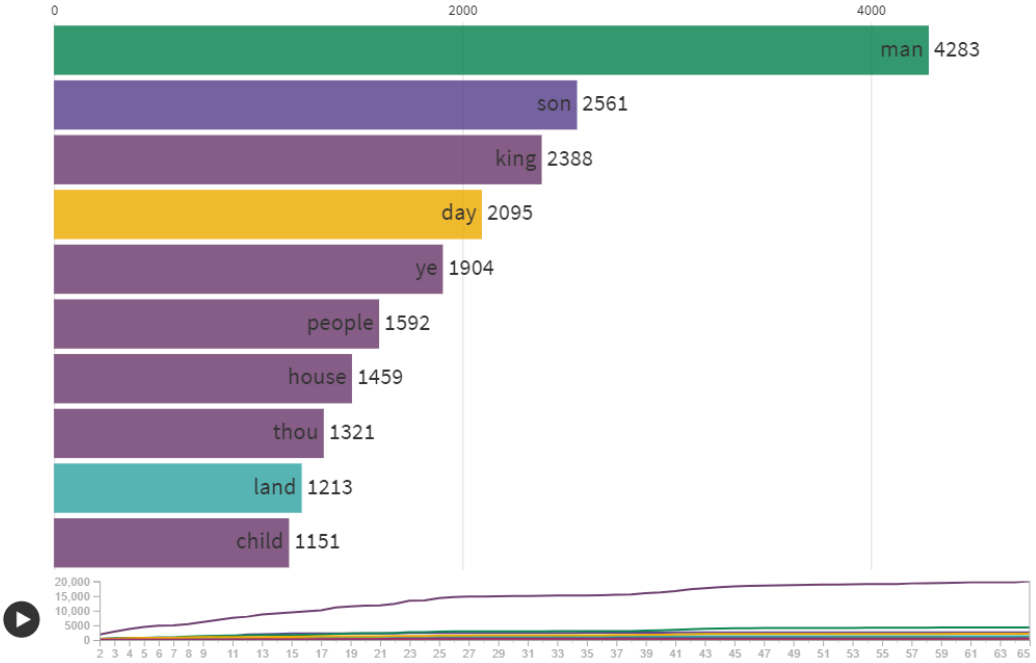


Figure 20: Most popular nouns in the Bible

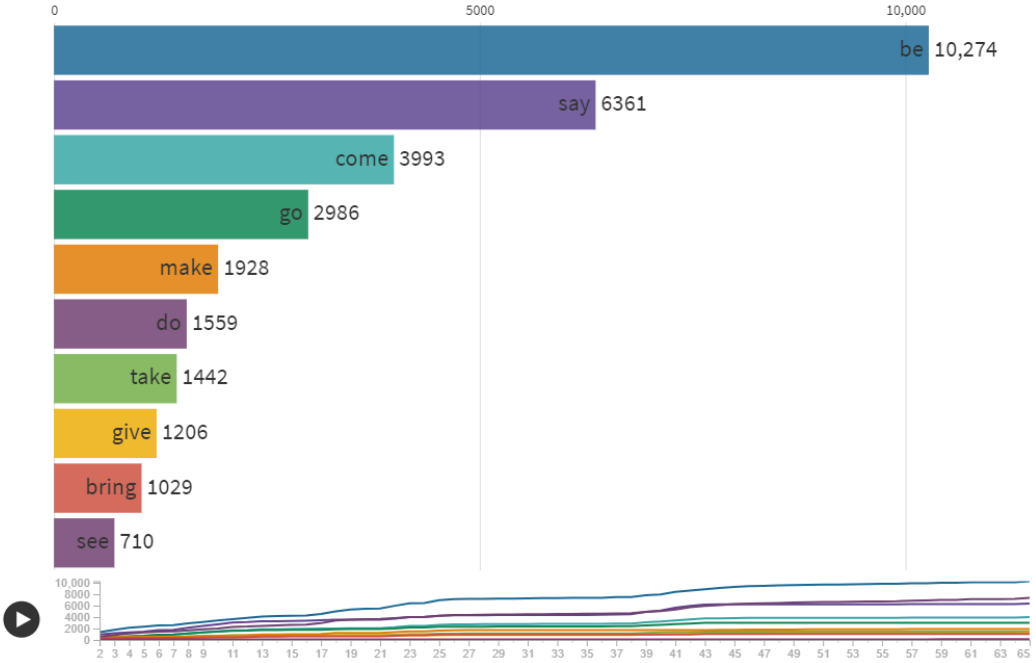


Figure 21: Most popular verbs in the Bible

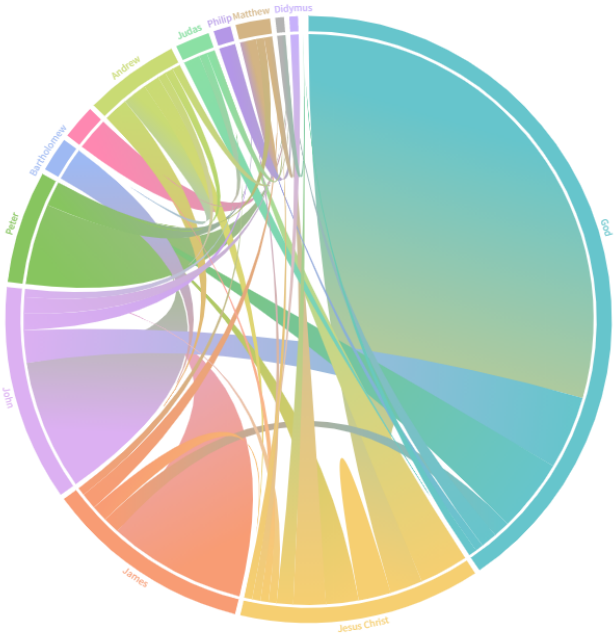


Figure 22: Co-occurences of God, Jesus and his disciples through the New Testament.

References

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 annual conference of the north american chapter of the association for computational linguistics (demonstrations)* (pp. 54–59).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding..

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

Hutto, C., & Gilbert, E. (2014). *Vader: A parsimonious rule-based model for sentiment analysis of social media text.* GitHub.

Loria, S. (2020). *Textblob.* `https://github.com/sloria/TextBlob`. GitHub.

Mendels, O. (May 8 2020). What is it with nlp models and biblical names?

Webster, J. J., & Kit, C. (1992). Tokenization as the initial phase in nlp. In *Coling 1992 volume 4: The 15th international conference on computational linguistics.*

Ye, Z., Jiang, G., Liu, Y., Li, Z., & Yuan, J. (2020). Document and word representations generated by graph convolutional network and bert for short text classification.