## Introduction

The goal of this project is to find and test different approaches in order to be able to recommend articles form one article. If the user is reading an article, then the app should be able to recommend some other articles related to the subject of the original article. Having that in mind, it is noted that the implementation of the approaches should be relatively short in terms of calculating time. The user should not need to wait for a prolonged period before receiving new article recommendations. Knowing also that the calculation time is heavily reliant on the number of articles, but also on the way each approach is implemented. For example, relatively high computing time when initializing to set up the model and to pre-process all the data will not affect the user experience because this process is done at an earlier time, with the results stored. The desired outcome is that requesting related articles does not take too long and the computation time when adding a new article stays within acceptable limits. In order to achieve the task of finding recommending articles, techniques for topic modelling and scoring similarities between articles were researched on. The three techniques selected are using Term Frequency–Inverse Document Frequency (TF-IDF), Latent Dirichlet Allocation (LDA) and sentence embeddings. These three methods suppose that the data is cleaned and pre-processed beforehand, so part of this report focuses on the pre-processing of the data then it is all about the two approaches used and their results.

## Pre-processing

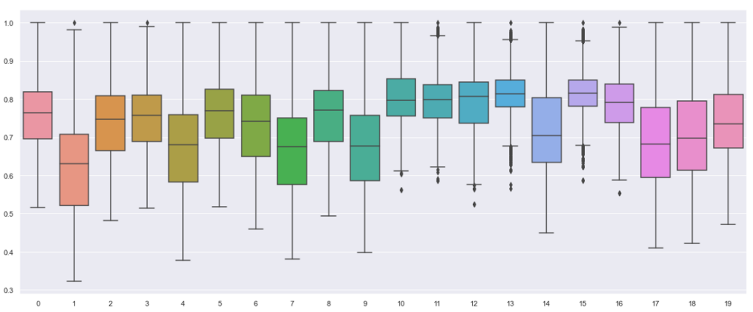
In NLP, preprocessing and data cleaning is an important part of the process. The data consisted of a list of articles with their titles.   
The first step was to clean the data. The formats of the articles were different and some of them had html tags. So, the first step was to remove them using regex. Then, remove characters not recognized due to encoding issues, such as “;” and “&”. Any numbers were also removed because finding number matches between articles are not significant in finding relevant articles.

With the data cleaned, the data is then pre-processed by removing stop-words and lemmatizing the text.  
Stop-words are common words in a designed language. Stop-words were obtained from the nltk library to remove common Swedish words that do not add much meaning to a sentence. Having the stop-words present in the data will cause the results to be inaccurate given the frequent use of these stop-words in every text. If stop-words are not removed, articles become too similar only because they are written in the same language and thus having the same recurrent stop-words.  
Lemmatization is the process of identifying the “root word” for any given word. For example, if we have a verb we want to remove any conjugation and have the verb on its simplest form. For nouns and adjectives, we want to remove any plural forms etc.… So, if you apply that to our articles we get better results because words that didn’t match because of one being in the plural form or because of the conjugation, now match. To implement lemmatization there are multiple solutions, we use the *spaCy* library, very frequently used for NLP purposes.

## Latent Dirichlet Allocation (LDA) approach

The Latent Dirichlet Allocation is an algorithm used for topic modelling which consider a document as a mix of topics and each topic as a mix of words. So, the algorithm scores each word for each topic. So basically, the algorithm says for each word how probable it is that it belongs to each topic. We then can represent each topic with the top X words. The problem with this method is that each document is viewed as a collection of words, so the grammatical order of the words does not matter. We must make sure that the stop words are well removed otherwise it will bias the results. One of the other problems is that we have to know how many different topics we want cause it’s a parameter of the model.

## TF-IDF approach

The next step we have is tokenization and sentence embedding. Because we can’t pass words into a model we must use numbers. So basically, this step is translating words into number. The basic solution to solve that issue is to one-hot encode every word of each article, so you’ll end up with a list of numbers corresponding to the mapping of each word. The downside of this method is that any relation between words is omitted as well as the disposition of each word. To counter this downside, we have another method called words embeddings. This method can be implemented using a pretrained model like Word2Vec developed by Google. It maps each word to a vector of numbers, so that each word that a like between each other like have somehow a quite similar vector. For example, if we map the name of two countries like “Sweden” and “Denmark” then both words vectors should be close to each other. This solution solves the problem of relation between words. But it stills doesn’t consider the whole sentence because it processes each sentence individually. We tested the TF-IDF method, it goes through all the documents and creates a matrix corresponding each word to each document. It assigns a weight to each pair word/document. The weight increases if this word appears often in the document and it decreases if the frequency of the word among all documents. We used this method in two ways. The first way to pass it the documents pre-processed. The second way is to pass the result of the named entity recognition models. The named entity recognition model we use is in the *spaCy* library with one of their pre-trained Swedish language models and we pass it our text. We use sentence embedding to be more accurate on conversion from text to numbers and to get all the relations between words in a sentence. There are multiples methods to embed a sentence. We are only going to talk about the one we tested. Unlike words embeddings, sentence embeddings give a unique vector based on the sentence. For example, with these two sentences: “Eleven players are on the field.” & “He is the best player in the team”, even though player has the same meaning in this case they will not have the same representation. This pre-trained model is able to detect the function of each word in the sentence, to tell whether it’s a noun, an adjective, a verb… And it also can tell the words related to it in the sentence. With all these information its able to provide us with a vector of numbers representing the whole sentence. Using that we can see which document looks like another one by using cosine similarity because we have an embedded vector for each article. The cosine similarity ranges from 0 to 1. The higher the cosine similarity is the higher the similarity between each article is.

## Sentence Embeddings

This approach explores the utility of pre-trained models. Given relatively little data, using pre-trained models might provide better and more efficient results as large amounts of data was trained in the model. The pre-trained model used for this approach is the swedish-bert-based-cased pre-trained model because Swedish articles is provided. Using the cleaned and pre-processed data, word embeddings are obtained from the pre-trained model followed by sentence embeddings. The sentence embeddings are calculated by averaging the word embeddings in each article. It should be noted that lemmatization was not performed for this approach as it causes the removal of context in each article. As the pre-trained models already contains the different word forms, lemmatization can be omitted. Cosine similarity scores are also used to score the similarities between two sentence embeddings representing the original articles. To evaluate the accuracy of the model, given links data is used. The desired outcome is that tagged articles should have a high cosine similarity score between each other, as the tagging was done to group similar articles together. If the tagged articles should have a high cosine similarity score in the current model, it would show that the model is working as per intended.

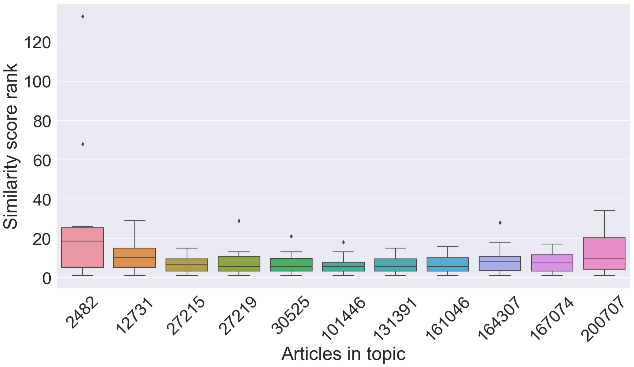
The approach to evaluate the accuracy of the model is as follows:

1. Sort the tags by frequency in the given links data
2. For each tag:
   1. Choose the first article found having the same tag
   2. Average the cosine similarity scores for articles with the same tag using the first article as basis of comparison
      1. Articles with the same tag should have a high cosine similarity score. This means that the percentile of the cosine similarity scores should be high as well
   3. Calculate percentile of averaged cosine similarity score
3. Average the percentiles obtained for each tag

The resultant percentile obtained is 93.33. This means cosine similarity scores between tagged articles are in the 93rd percentile on average, implying the cosine similarity scores between tagged articles are relatively much higher than the remaining articles that do not have the same tag. As such, it can be concluded that cosine similarity scores from the sentence embeddings can be used to rank and recommend similar articles to a large extent.

## Approaches Evaluation & Conclusion

We faced a big problem in evaluating our different approaches, and this issue applies to any approaches we could have used. Is the lack of labeled data. At first the only tags we had were very heterogenous, that is to say many of them were irrelevant. Many articles had tags based on the author and we do not want our models to think articles are similar because it’s the same journalist who wrote them. So, because of this unlabeled data we had to come up with a method to evaluate our models. Our decision was to use the cosine similarity we talked about before. So, we take a sample of articles from different topics, and we evaluate their similarity with the other articles from the same topic. To do so we take the average cosine similarity for every article in the topic. The downside of this method is because we do not compare every article in the topic between one another. We only take a sample of the articles to evaluate the results so when we take a sample we might have luck and take only articles that have good cosine similarity within their topic. We then come up with the following boxplot:   
The graphic show us that for every topic. If we pick a random article then its cosine similarity with 75% of the group is not as low as 0,5.

The TF-IDF gave us pretty comparable results according to the following graphic, the method used to get it is the same as the previous graphic:

Chart, waterfall chart

Description automatically generatedOn this graphic, lower is better. On this case we use the full text whereas on the graphic below with used named entity recognition which we talked about before and is about getting the links between words in a sentence.

On this graphic as well, lower is better. And we can see that for a particular topic, the named entity recognition does not do a really good job compared to the full text surprisingly.  
The next step that could really help us making a big step on the different approaches’ evaluation, is succeeding to the implement a systemic test approach using the related articles provided by Linus. That could help us have a quantitative approach regarding the evaluation of our models. Another way to go forward would be to find a dataset of news articles written in Swedish already labelled by categories for example (“politics”, “sports” etc.…) and to train a model using those data and then evaluate the created model with the related data gave by Linus to see if this approach performs better than the ones we tested and implemented.