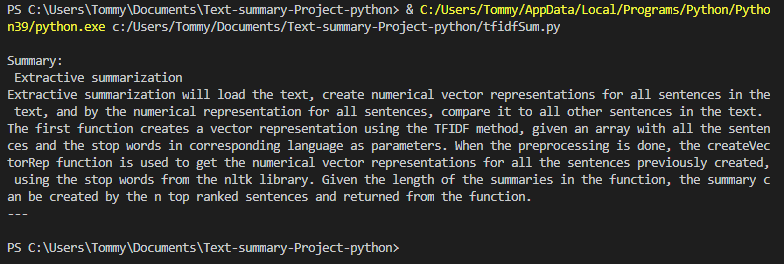
Text summarizing

Natural language processing

By: Tommy Granström, tomgr497



*Summary of the document made by the program.*

**Table of contents:**

[Problem statement 1](#_Toc62581112)

[To solve this problem 1](#_Toc62581113)

[Abstract summarization 1](#_Toc62581114)

[Extractive summarization 1](#_Toc62581115)

[Choosing approach 1](#_Toc62581116)

[Theory 2](#_Toc62581117)

[Create a numerical vector representation 2](#_Toc62581118)

[TFIDF - score 3](#_Toc62581119)

[Data: 3](#_Toc62581120)

[Implementation: 4](#_Toc62581121)

[Evaluation 7](#_Toc62581122)

[Appendix 8](#_Toc62581123)

[Text 1 8](#_Toc62581124)

[Text 2 9](#_Toc62581125)

[Text 3 10](#_Toc62581126)

[Text 4 11](#_Toc62581127)

[Text 5 12](#_Toc62581128)

# Problem statement

Everyday millions of articles are published by both newspapers and overall different websites. By all these articles, some of them you come across by browsing social media, favorite newspaper or just by being recommended by a friend or colleague. If only reading the ones you find interesting, there is still a lot of them, and it would be very time consuming.

# To solve this problem

To solve this problem, I decided to develop a text summarizer, where the idea was to load the text from an interesting article (English or Swedish) and get it automatically summarized. Solving this problem, there is two approaches. Either **abstract** summarizationor **extractive** text summarization.

## Abstract summarization

Abstract summarization is when the program itself loads the data, create a numerical representation of the text and then predict a numerical summarization, which then is decoded as text. In other words, it reads the text and create a summary with “its own words”. To learn about the abstract method, I read and followed the tutorial in this article:

<https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/> by Pai Aravind.

## Extractive summarization

Extractive summarization will load the text, create numerical vector representations for all sentences in the text, and by the numerical representation for all sentences, compare it to all other sentences in the text. The resulting summary will be the *n* sentences that are the most similar to all other sentences, where *n* is specified by the user to set how long the summary should be. To learn about the extractive approach, I read and followed the tutorial in this article:

<https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/> by Joshi Prateek.

## Choosing approach

Abstract summarization would be able to create more creative and extensive summaries compared to an extractive method. Although, using an abstract method will demand a lot of training data and the result would probably be affected by the chosen type of training data. Where the training data would be a data set of different texts with corresponding summaries. Possible errors and bugs of an abstractive method could also result in misleading and incorrect summaries.

Extractive summarization would always guarantee a relevant summary considering only the sentences from the given text is the base when building the summary. Although, an extractive method could result in an incorrect summary considering the grammar but will probably at least contain the fundamental information of the given text.

An extractive method will only use the given text and stop words in corresponding language as input, which makes it ready to summarize all types of texts independent of type or length of the text. An abstract method will need to be trained for the type of texts it will summarize, to make good summaries.

Considering these pros and cons of both methods, I choose to use the extractive approach.

# Theory

After reading Parteeks article I got the idea of how the extractive summarizing method works, and how I want to model my text summarizer. The extractive summarizer method works as follows:

Given the text you want to summarize you will:

* Do some basic preprocessing such as: remove stop words, special characters and make all character lower case.
* Divide the text into sentences, so you are given an array with all the sentences.
* Create a numerical vector representation for each sentence.
* Given the numerical representation for each sentence, compare to all other sentences.
* Rank the sentences, where the top ranked sentence is the one that is most similar to all other sentences.
* Build a summary of the top n ranked sentences, by adding them together. Where *n* determines how long the summary should be.

Before going further to the implementation some of the steps needs to be further explained.

## Create a numerical vector representation

Reading the article, Parteek used a pretrained dataset to create the vector representations, named *GloVe* (GloVe: Global Vectors for Word Representation). Where the dataset can be described as a dictionary, where each word has its own predefined vector representation, with 100 dimensions. More about this dataset can be found here: <https://nlp.stanford.edu/projects/glove/>

Using this dataset, which has 100 dimensions per word and 400 000 different words will be very time memory consuming to load as a dataset but also time consuming when building the vector representation, considering the time complexity to get the vector representation for each word in the dictionary.

Instead of using this predefined dataset, I decided to use the Text frequency Inverse document frequency – score (TFIDF - score) method to build my vector representations. This method is further described below.

## TFIDF - score

To create a vector representation of the sentences, the TFIDF method is used. The vector representation of a sentence will be a vector where each dimension corresponds to the frequency of each word from the dictionary in the sentence. Given the frequency vector, the inverse vector is calculated, hence the most empirical informant words are widely assumed to be the ones least mentioned. The inverse frequency vector will be the vector representation.

The comparison to all other sentences is then done by computing the cosine similarity between each sentence(vector) to all other sentences and sum the total angular difference. The Lowest sum of angular difference is the sentence most alike all other sentences.

More about the TFIDF-score in Lab 4 in the TNM108 course and in this article:

<https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089> By: William Scott.

# Data:

To solve this, no data is needed to train the model since the extractive method only considers the given text when summarizing. Although when summarizing a text there is no right answers, but to get independent examples I downloaded the data set *News Summary* from: <https://www.kaggle.com/sunnysai12345/news-summary> .

This data set contains about 4000 texts that have been summarized by the Inshort news app. For the evaluation I choosed three random texts.

I also used two Swedish text from Wikipedia, the basic definition of Machine learning in Swedish:

<https://sv.wikipedia.org/wiki/Maskininl%C3%A4rning#:~:text=Maskininl%C3%A4rning%20(engelska%3A%20machine%20learning),regler%20f%C3%B6r%20just%20den%20uppgiften>.

And Norrköping history:

<https://sv.wikipedia.org/wiki/Norrk%C3%B6pings_historia>

# Implementation:

The text summarizer is created using Python. Some special libraries are required for this project, so I started with installing these Python Libraries:

* sklearn
* numpy
* nltk
* network

Then I started by create two functions. The first function creates a vector representation using the TFIDF method, given an array with all the sentences and the stop words in corresponding language as parameters. This function is named *createVectorRep.* Seen below:

def createVectorRep(textCentenses,stopwords):

    #Create vocabulary

    my\_vocabulary = []

    from nltk.tokenize import word\_tokenize

    for i in textCentenses:

        my\_vocabulary.append(word\_tokenize(i))

    my\_vocabulary = [y for x in my\_vocabulary for y in x] # flatten list

    my\_vocabulary = list(set(my\_vocabulary))

    #Create your vectorizer

    from sklearn.feature\_extraction.text import CountVectorizer

    vectorizer = CountVectorizer(stop\_words=stopwords,vocabulary=my\_vocabulary )

    #Create tfidf vector rep

    smatrix = vectorizer.transform(textCentenses) #Sparse matrix

    from sklearn.feature\_extraction.text import TfidfTransformer

    tfidf\_transformer = TfidfTransformer(norm="l2")

    tfidf\_transformer.fit(smatrix)

    tf\_idf\_vectortt = tfidf\_transformer.transform(smatrix)

    tf\_idf\_vector = tf\_idf\_vectortt.todense()

    return tf\_idf\_vector

The first thing to do in the *createVectorRep* function is to create a dictionary of the words in the sentences using the *word\_tokenizer* from the *nltk* library. When I have created the dictionary and have the stop words in corresponding language, we can now create an instance of the *CountVectorizer* from *sklearn* with stop words and the dictionary as parameters.

The *CountVectorizer* will be the frequency vectors for all the sentences. We can then create the numerical vector representation by creating an instance of the *TfidifTransformer* from *sklearn* which will transform the frequency vectors to TFIDF vector representations of the sentences. Which will be the returned object from the *createVectorRep* function.

Next function to create is a function to make the actual summary, using the *createVectorRep* function. The function to create the summary is named *makeTextSummary* and takes the text to summarize, language and length of summary as parameters.

def makeTextSummary(text,textLanguage,sl):

    #Pre processing of raw text

    sentences = []

    from nltk.tokenize import sent\_tokenize

    sentences.append(sent\_tokenize(text)) #Sentences tokenize

    sentences = [y for x in sentences for y in x] # flatten list

    clean\_sentences = [s.lower() for s in sentences] #Lower case ü

    clean\_sentences = pd.Series(clean\_sentences).str.replace("ü","u") #If german

    clean\_sentences = pd.Series(clean\_sentences).str.replace("å", "a") # if Swedish

    clean\_sentences = pd.Series(clean\_sentences).str.replace("ä", "a")

    clean\_sentences = pd.Series(clean\_sentences).str.replace("ö", "o")

    clean\_sentences = pd.Series(clean\_sentences).str.replace("?", " ")

    clean\_sentences = pd.Series(clean\_sentences).str.replace("[^a-zA-Z]", " ") # Remove special charachters

    from nltk.corpus import stopwords

    stop\_words = stopwords.words(textLanguage) #Load stop words for corresponding language

    from sklearn.metrics.pairwise import cosine\_similarity

    import networkx as nx

    sentence\_vectors1 = createVectorRep(clean\_sentences,stop\_words)#TFIDF

    # similarity matrix

    sim\_mat1 = np.zeros([len(sentences), len(sentences)])

    for i in range(len(sentences)):

        for j in range(len(sentences)):

            if i != j:

                sim\_mat1[i][j] = cosine\_similarity(sentence\_vectors1[i], sentence\_vectors1[j])[0,0]

    nx\_graph1 = nx.from\_numpy\_array(sim\_mat1)

    scores1 = nx.pagerank(nx\_graph1)

    ranked\_sentences1 = sorted(((scores1[i],s) for i,s in enumerate(sentences)), reverse=True)

    return summary

The makeTextSummary first does the preprocessing of the text. Using the *sent\_tokenize* from the *nltk* library it extracts all sentences from the text and create an array with all the sentences. Next step is to replace all special characters, and then make all characters lowercased.

When the preprocessing is done, the *createVectorRep* function is used to get the numerical vector representations for all the sentences previously created, using the stop words from the *nltk* library.

Given the vector representation, a similarity matrix is created containing the cosine similarity between each pair of sentences. The similarity matrix is then converted to a graph using the *networkx* library. Given the dense graph, the *pageRank* function from the *networkx* library can be used to rank the sentences. The *pageRank* function will assume the highest value is the most likely sentences to describe all other sentences, but since cosine similarities is used to represent the similarity, the value will be lower the more similar the sentences are. Therefore, the ranked sentence must be sorted in reverse. Given the length of the summaries in the function, the summary can be created by the *n* top ranked sentences and returned from the function. More about the PageRank algorithm can be found in Parteeks article.

When the two functions have been created, we can now load the texts we want to summarize from the dataset earlier mentioned. When the texts have been loaded, the summaries can be created.

#Load article dataset

texts = []

summaries = []

with open("sumTexts.csv") as f:

    lines = f.readlines()

    for i in range(1,len(lines)):

        a = lines[i].split(";")

        texts.append(a[0])

        summaries.append(a[1])

for i in range(len(texts)):

    makeTextSummary(texts[i],'english',3)

The code for this project can be found in my Github repo (tfidfSum.py):

<https://github.com/tommygranstrom/Text-summary-Project-python>

# Evaluation

The texts and corresponding summaries can be seen in Appendix. Since no right answer exist on how to summarize a text, a user evaluation was done to evaluate how good the summarizer works.

The user evaluation contained the English texts from the downloaded dataset. For each text, the summary from the Inshort news app, the summary using the Glove method and the TFIDF-method was given. The users were asked to rate each summary from 1-5.

To test the ability to summarize in other languages, the Swedish texts copied from Wikipedia was also summarized using the TFIDF method and given for the user to rate 1-5.

9 Users evaluated the summaries (not all users rated each summary). In figure 1 the result for the three English texts can be seen. (Inshort news = blue, TFIDF = red, GloVe = Yellow)

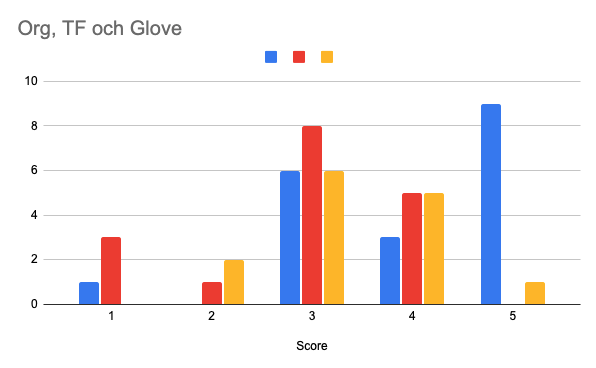


Figure 1 – Frequency of each score for the three types of summaries

Further, the two Swedish texts got the average score of 3.15. As seen in figure 1, the Glove method and Inshort news got the best result, but majority of the TFIDF-method still got rated to 3 or higher. Looking at the summaries from the TFIDF - method, one can draw the conclusion that the grammar is not always right but does at least contain the fundamental information from the text it summarized.

As a bonus, a single case evaluation of three German articles was done with the same conclusion, that it contained the fundamental information, but incorrect grammar.

In conclusion, the TFIDF summarizer can summarize and give the user the fundamental information about the text, but the grammar will not always be correct. The Inshort news - and Glove methods, gave a bit better score by the users. Although the TFIDF method could be considered good enough since its flexibility, “lightweight implementation” and ability to at least generate feasible summaries.

# Appendix

In this section the text for evaluation with corresponding summaries using the Original Inshort news summary, Glove-method and TFIDF – method is presented.

## Text 1

**Text to summarize:**

“Two years after a helicopter crash near the Bombay High offshore oil field killed two pilots, an inquiry by the Air Accident Investigation Bureau (AAIB) found that the chopper was flying at a critically low height, without the pilots realising it. The Pawan Hans helicopter was on a training sortie on the evening of November 4, 2015. Aviation regulator Director General of Civil Aviation (DGCA) had asked AAIB to conduct an inquiry. The inquiry report, which was made public on Wednesday, said that the main pilot was flying low and that the other did not realise it. After taking off, the Rontappmeyer (RYTYR) helicopter may have faced difficulties as the moon was waning, the report said. The report mentioned that the instructor, Captain E Samuel, wasn't disoriented and was aware of the low height of the helicopter. He alerted Captain TK Guha a little late, but even after realising that Guha wasn't able to manage the situation, Samuel did not take control of the helicopter. AAIB recommended that ONGC, which owns Bombay High along with the helicopter operators, identify helidecks from where night flying can take place. AAIB also said that Pawan Hans pilots need to rest sufficiently before flying. Guha was flying during the night after a gap of almost a year, read the report. READPawan Hans chopper crash: Both pilots on board still missing.”

**Inshort news summary**:

An inquiry by the Aircraft Accident Investigation Bureau found that the Pawan Hans helicopter was flying at a critically low height, resulting in its crash near Bombay High in 2015. The report further mentioned that the instructor, despite being aware of the low height, didn't take control of the helicopter after the trainee pilot wasn't able to manage the situation.

**TFIDF Summary**:

AAIB also said that Pawan Hans pilots need to rest sufficiently before flying. Two years after a helicopter crash near the Bombay High offshore oil field killed two pilots, an inquiry by the Air Accident Investigation Bureau (AAIB) found that the chopper was flying at a critically low height, without the pilots realising it.

**Glove Summary**:

The inquiry report, which was made public on Wednesday, said that the main pilot was flying low and that the other did not realise it. Two years after a helicopter crash near the Bombay High offshore oil field killed two pilots, an inquiry by the Air Accident Investigation Bureau (AAIB) found that the chopper was flying at a critically low height, without the pilots realising it.

## Text 2

**Text to summarize**:

“Only 26 malls and commercial centres on Mehrauli-Gurgaon (MG) road, Golf Course Road and Sohna Road will offer free parking in their basements, the Municipal Corporation of Gurugram (MCG) announced on Wednesday. The MCG had earlier said it will waive property tax on basements if the same were used to provide free parking. Soon afterwards, 45 sites offered free parking but now some of the owners have agreed to pay the property tax instead of offering free parking service. As a result, the list has been trimmed to 26. Building owners said the revenue from parking is higher than the gains from the property tax waiver.The updated list is in the public domain on the civic body's website : www.mcg.gov.in. As reported earlier, MCG officials had reviewed the Central governments notification regarding evaluation of property tax for commercial spaces and that to led to the civic body announcing the clause of tax waiver in lieu of free parking space.The 26 establishments offering free parking are Suncity Business Tower, Paras Twin Towers, MGF The Palm Springs Plaza, Global Foyer, Vipul Tech Square, MPD Tower, Augusta Point, Paras Downtown Tower, Central Plaza and Centrum Plaza : all located on the Golf Course Road. On MG Road, ABW Tower, Sewa Corporate Park, Garden City Point, DLF Corporate Park, Platina Mall and Vipul Agora Mall will offer free parking. Unitech Cyber Park Sector 39, Unitech Business Zone and Nirvana Country Yard Sector 50 and Unitech Arcadia Sector 49 will also offer free parking.Omaxe City Centre, Omaxe Mall, Raheja Mall, ILD Trade Centre, Omaxe Celebration Mall and the Sapphire Mall on Sohna Road are also on the list. The MCG had the option of levying a property tax on the basements in commercial centres, but after reading the Central government guidelines, an option was given to building owners that if they did not levy a parking fee on customers, the property tax on basements will be waived. Hence, for the publics benefit, the MCG has placed free parking boards at these sites for their awareness MCG Commissioner V Umashankar said. If the sites mentioned on the list are found levying a parking fee, the public can report complaints on the MCG's toll-free number 1800-180-1817 or email to support@mcg.gov.in.”

**TFIDF Summary:**

Only 26 malls and commercial centres on Mehrauli-Gurgaon (MG) road, Golf Course Road and Sohna Road will offer free parking in their basements, the Municipal Corporation of Gurugram (MCG) announced on Wednesday.The MCG had earlier said it will waive property tax on basements if the same were used to provide free parking.

MCG Commissioner V Umashankar said.If the sites mentioned on the list are found levying a parking fee, the public can report complaints on the MCG?s toll-free number ?

**Inshort news Summary:**

Municipal Corporation of Gurugram on Wednesday said that 19 out of 45 commercial building owners have decided to pay property tax instead of providing free parking to the public. Notably, MCG earlier offered a property tax waiver for building basements if they were used to provide free parking. However, the owners allegedly said that revenues from parking profited them more.

**Glove summary**:

Only 26 malls and commercial centres on Mehrauli-Gurgaon (MG) road, Golf Course Road and Sohna Road will offer free parking in their basements, the Municipal Corporation of Gurugram (MCG) announced on Wednesday.The MCG had earlier said it will waive property tax on basements if the same were used to provide free parking. As reported earlier, MCG officials had reviewed the Central government's notification regarding evaluation of property tax for commercial spaces and that to led to the civic body announcing the clause of tax waiver in lieu of free parking space.The 26 establishments offering free parking are Suncity Business Tower, Paras Twin Towers, MGF The Palm Springs Plaza, Global Foyer, Vipul Tech Square, MPD Tower, Augusta Point, Paras Downtown Tower, Central Plaza and Centrum Plaza.

## Text 3

**Text to summarize**:

“New Delhi, Aug 2 (PTI) Bomb squads and canine teams were today rushed in to check a suspect object that was recovered at the cargo hold area of the IGI airport here, later declared safe after it was found that the consignment only contained some auto spare parts. CISF Director General O P Singh said the "suspect item has been declared safe and there is nothing to worry about". At 7.15 am, an X-ray image of a consignment of Maruti spare parts raised suspicion among the staff of the domestic cargo terminal, according to sources in the Bureau of Civil Aviation Security (BCAS). The terminal staff immediately alerted the CISF, who along with a team of BCAS and a bomb detection and disposal squad rushed to the spot. “

**Glove summary**:

New Delhi, Aug 2 (PTI) Bomb squads and canine teams were today rushed in to check a suspect object that was recovered at the cargo hold area of the IGI airport here, later declared safe after it was found that the consignment only contained some auto spare parts. At 7.15 am, an X-ray image of a consignment of Maruti spare parts raised suspicion among the staff of the domestic cargo terminal, according to sources in the Bureau of Civil Aviation Security (BCAS).

**TFIDF Summary**:

New Delhi, Aug 2 (PTI) Bomb squads and canine teams were today rushed in to check a suspect object that was recovered at the cargo hold area of the IGI airport here, later declared safe after it was found that the consignment only contained some auto spare parts. The terminal staff immediately alerted the CISF, who along with a team of BCAS and a bomb detection and disposal squad rushed to the spot.

**Inshort news summary**:

Bomb squads and canine teams were rushed to check a package recovered in the cargo hold area of Delhi airport on Wednesday. Officials later declared the area safe, adding, "the suspect item was nothing but some spare parts of company Maruti." Its X-ray image had raised suspicion among airport staff.

## Text 4

**Text to summarize:**

Norrköpings historia som stad började 1384 när orten fick sina stadsrättigheter bekräftade. Då hade människor redan bott kring Motala ströms fall under lång tid. Staden fick sin första storhetstid under 1600-talet när den holländske affärsmannen Louis de Geer slog sig ner i staden. Han grundade ett flertal industrier i staden, bland annat ett pappersbruk, ett vapenfaktori, en klädesfabrik och ett skeppsvarv. Under denna tid var staden Sveriges till folkmängden näst största stad. Under 1800-talet växte textilindustrin i Norrköping och hälften av klädestillverkningen i Sverige skedde i Norrköping. De goda tiderna varade ända fram till 1950-talet då konkurrensen från utlandet började kännas av. Konkurrensen blev för svår och 1970 fanns bara ett fåtal textilfabriker kvar. I stället flyttade flera statliga verk till staden.

**TFIDF summary**:

Staden fick sin första storhetstid under 1600-talet när den holländske affärsmannen Louis de Geer slog sig ner i staden. Under denna tid var staden Sveriges till folkmängden näst största stad.

## Text 5

**Text to summarize**:

”Maskininlärning (engelska: machine learning) är ett område inom artificiell intelligens, och därmed inom datavetenskapen. Det handlar om metoder för att med data träna datorer att upptäcka och lära sig regler för att lösa en uppgift, utan att datorerna har programmerats med regler för just den uppgiften. Området angränsar till statistik, datorseende och mönsterigenkänning. Datautvinning (datamining) är ett betydelsenära begrepp som avser en kombiniation av maskininlärning och statistiska metoder för att upptäcka och visualisera mönster i stora mängder data. Maskininlärningsmetoder arbetar med data. Ofta kan datan delas in i indata (den information som datorn får tillgång till) och utdata (det svar som datorn förväntas ge efter att ha bearbetat indatan). En definition av maskininlärning har skapats av Tom M. Mitchell[1]. Han definierar de algoritmer som studeras inom maskininlärning som ett datorprogram sägs lära av erfarenhet E givet en uppgift T och ett prestandamått P om dess prestanda på uppgiften T, mätt med P, ökar med erfarenhet E.”

**TFIDF Summary**:

Datautvinning (datamining) är ett betydelsenära begrepp som avser en kombiniation av maskininlärning och statistiska metoder för att upptäcka och visualisera mönster i stora mängder data. Han definierar de algoritmer som studeras inom maskininlärning som ett datorprogram sägs lära av erfarenhet E givet en uppgift T och ett prestandamått P om dess prestanda på uppgiften T, mätt med P, ökar med erfarenhet E.