|  |  |  |
| --- | --- | --- |
|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика |  |

**Съвместен курсов проект по**

**Подходи за обработка на естествен език**

**Извличане на информация**

на тема: „Резюмиране на текст с LSA и TextRank”

Студент: **Тервел Радославов Вълков,** Ф.Н. **9MI3400058**

Курс: 1, Учебна година: 2021/22

Преподаватели:

проф. Галя Ангелова, гл. ас. Ивелина Николова-Колева

=================================

<https://github.com/trvulkov/NLP-IE-project>

**Съдържание**

[1 Увод 2](#_Toc536441759)

[2 Подходи към резюмирането на Текст 2](#_Toc536441760)

[3 Проектиране 4](#_Toc536441761)

[3.1 Анализ на изискванията 4](#_Toc536441763)

[3.2 Обща архитектура 4](#_Toc536441764)

[3.3 Потребителски интерфейс 5](#_Toc536441764)

[4 Реализация, тестване/експерименти 5](#_Toc536441762)

[4.1 Обяснение на алгоритмите 5](#_Toc536441763)

[4.2 Използвани технологии, платформи и библиотеки 6](#_Toc536441763)

[4.3 Подготвяне на данните 7](#_Toc536441764)

[4.4 Реализация 8](#_Toc536441764)

[5 Заключение 11](#_Toc536441765)

[6 Използвана литература 12](#_Toc536441766)

# Увод

Резюме е кратко изложение, в което са изведени основните положения на някакъв текст. Може да се резюмират най-различни текстове – статии (научни, новинарски), книги и други художествени текстове, доклади, ревюта и други потребителски мнения, коментари, и т.н. Има множество приложения на резюметата - представяне само на най-важните части от текста, на сложна информация в кратък и стегнат формат, по-ефективна комуникация на важна информация (например при новини) и бързо определяне на тематиката на даден текст (особено полезно при научни статии). При художествените текстове, с резюмето може да се даде общ преглед върху сюжета, представящ основните моменти, с цел да се заинтересува нов читател да прочете творбата, или да се освежи паметта за вече прочетен текст. Чрез резюмирането на потребителски мнения, ревюта и коментари пък може да се спомогне за взимане на решения за даден продукт.

Резюметата обикновено се правят ръчно, от човек, който трябва да прочете текста и да реши кои детайли да включи и изключи, и как да ги изрази. Чрез автоматизирането на процеса, така че това да бъде извършвано от машина, може по-бързо и ефективно да получаваме резултат (но в зависимост от ползвания алгоритъм, може качеството на машинното резюме да е доста далеч от ръчно-направеното такова).[1]

# Подходи към резюмирането на текст

Съществуват две общи категории на подходи за автоматичното резюмиране[1][2]:

* *Extraction-based* (екстрактивни, базирани на извличане) – това са по-простите подходи, при които само се извлича съдържание от оригиналния текст, без да бъде модифицирано. Обикновено се взимат най-важните фрази или редове от текста, и така резултатното резюме е негово подмножество, и не включва нищо ново – възможно е само директно да се цитират неща вече съдържащи се в оригинала, липсва възможност за изразяване на детайли по друг начин. Поради това, резултатът може да изглежда по-схематичен и да не звучи особено като естествен текст.
* *Abstraction-based* (абстрактивни, базирани на абстракция) – по-сложните подходи, при които се изгражда вътрешно семантично представяне на съдържанието, чрез което е възможно да се създаде резюме по-близко до това което би направил човек – с възможност за перифразиране и генериране на нови изречения, а не само директен цитат. Обикновено са базирани на дълбоко обучение. Цената на това по-добро качество на самото резюме обаче е че тези подходи са много по-сложни от изчислителна гледна точка – изискват обработка на естествен език, а често и дълбоко разбиране за тематиката на оригиналния текст (когато той е обвързан с някаква специална област на знание, например при научни текстове). Също така, екстрактивните подходи могат да се адаптират и за резюмиране на неща различни от текст (например изображения или видео), което при абстрактивните е много по-трудно.
* *Подпомогнато резюмиране* - за най-високо качество се комбинира софтуер и човешко усилие:
  + при *Machine Aided Human Summarization* (човешко резюмиране, подпомогнато от машина)екстрактивна система дава кандидати за извличане, след което човек ги анализира и решава кои да избере (като може и да ги редактира допълнително, като добавя или премахва текст)
  + при *Human Aided Machine Summarization* (машинно резюмиране, подпомогнато от човек) системата дава готов изход, а човек го обработва допълнително и редактира

В тези категории съществуват множество конкретни методи и алгоритми. Към екстрактивните методи спадат:[3]

* методи, базирани на *машинно самообучение*, които третират проблема като класификационна задача (изреченията се класифицират като участващи или не в резюмето), ползвайки техники като *Naive Bayes*, класификационни дървета, *SVMs* и др.
* методи, базирани на *графи*, които представят документите като графи, с възли представящи изреченията на текста. Обикновено се базират на алгоритъма *PageRank*, но приложен към текстове вместо уебстраници, с цел извличане на най-важни ключови думи – примери са *TextRank* и *LexRank*.
* *Latent semantic analysis/indexing (LSA/LSI)* – базиран на *Singular Value Decomposition*
* *Luhn’s Heuristic Method* – базиран на *TF-IDF*
* и др.

Към абстрактивните спадат:[4]

* *структурно-базирани*, които се базират на дървета, шаблони, правила, графи, онтологии, и др.
* *семантично-базирани,* които използват мултимодални семантични модели, модели на семантично представяне на текст, модели на семантични графи, и др.
* базирани на *рекурентни невронни мрежи* - *sequence-to-sequence RNNs*
* и др.

# Проектиране

## Анализ на изискванията

За целта на проекта са избрани екстрактивни подходи. При тях няма да има нужда от трениране, само от документи които да резюмираме. Налага се все пак предварителна обработка на документите – първо, алгоритмите работят с обикновен текст (низ), така че е нужно първо да се извлече само текстовото съдържание на документите, и второ, при обработката на текст е добра идея да се приложат някои техники за *preprocessing*, като токенизация, премахване на някои елементи (пунктуация, числа, стоп-думи и др.), нормализация на всичко до малки букви, *stemming*, *part-of-speech tagging*, *lemmatization*, и др.

След обработката на текста и получаването му в нужния формат, той се подава на алгоритъма, който извършва процеса по резюмиране. Като резултат получаваме нови текстове, съдържащи резюметата. След това, ако за избраните текстове също така има и готови, ръчно-изготвени резюмета, може те да се сравнят с резултатите от алгоритъма, и чрез дадена метрика да се изчисли точността му.

## Обща архитектура

В случая, имаме клас *Corpus*, който прочита документите и техните резюмета от диска и ги запазва като низове. Документите са в обикновен текстов формат, на английски език, като няма стриктни изисквания към форматирането им, единствено към директорната структура – те се намират в директорията “*corpus*”, на същото място като кода, като всеки отделен корпус от данни е в собствена поддиректория, съдържаща статиите (всяка в отделен файл), и директория “*summaries*”, съдържаща резюметата (отново всяко в отделен файл, еднакъв брой със статиите, с еднакви файлови имена – файл от “*summaries*” с име X е резюме на документа X). Освен пазенето на данните, *Corpus* също така извършва и оценяването на машинни резюмета чрез *ROUGE* метрики.

Функционалността по резюмиране се съдържа в отделни класове. Те са наследници на абстрактния *AbstractSummarizer*, съдържащ две функции, които следва да се имплементират в наследниците – *process\_text*, която трябва да резюмира само един текст, подаден директно като низ (и присъства основно за целите на тестване, не се прилага в крайната версия на проекта), и *process\_corpus*, която трябва да получава обект от класа *Corpus* и да резюмира всички негови документи, след което да се позове на функцията му за оценяване чрез *ROUGE*, и накрая или да покаже резултатите на конзолата, или да ги запише във файлове, намиращи се в нова поддиректория в директорията на корпуса (по един файл за всеки документ, съдържащ оригиналния текст, създаденото резюме, и оценката).

В проекта са включени два конкретни наследника – *TextrankSummarizer* и *LSASummarizer*, които извършват резюмирането съответно чрез алгоритмите *TextRank* и *Latent Semantic Analysis/Indexing* (всъщност *TextrankSummarizer* има и възможност за извършване на резюмиране чрез два други алгоритъма – *PositionRank* и *Biased TextRank*).

## Потребителски интерфейс

Програмата има прост потребителски интерфейс чрез командния ред:

summary.py [-h]

[-d {DUC, WikiHow, MultiNews}]

[-r RANGE RANGE]

[-t {textrank, positionrank, biasedtextrank, lsa}]

[-m {en\_core\_web\_sm, en\_core\_web\_md, en\_core\_web\_lg}]

[-s SENTENCES] [-w]

Като опциите са:

-h (--help) – съобщение за помощ

-d (--dataset) {DUC, WikiHow, MultiNews} – име на корпуса който искаме да резюмираме (трябва да е един от предварително подготвените в папката corpus)

-r (--range) a b – две числа, a < b, формиращи интервал (от поредни номера на документи в съответния корпус), в случай че искаме да резюмираме само под-последователност от корпуса

-t (--type) {textrank, positionrank, biasedtextrank, lsa} – алгоритъмът за резюмиране, който искаме да използваме

-m (--model) {en\_core\_web\_sm, en\_core\_web\_md, en\_core\_web\_lg} – в случая на TextRank алгоритмите, името на предварително тренирания модел, нужен за работата на избраната библиотека – три модела на английски, с различни размери (по-големите би трябвало да дават по-добри резултати, но удължават нужното време за изпълнение)

-s (--sentences) n – броя изречения, които искаме да съдържат резюметата

-w (--write) – дали да запишем резултатите на диска (вместо просто да ги отпечатаме на конзолата)

# Реализация, тестване/експерименти

## Обяснение на алгоритмите

Алгоритъмът *TextRank*[5] се базира на *PageRank* – алгоритъм на Google за ранкиране на уебстраници, който представя множеството от страници като ориентиран граф (с възли самите страници, и ребра линковете между тях), и за всяка от тях итеративно изчислява тегло чрез формула, отчитаща броя линкове към страницата (тоест на броя ребра към съответния възел) и теглата на страниците, които съдържат тези линкове (тоест възлите, от които излизат съответните ребра). Идеята на алгоритъма е, че страница към която има много линкове е по-„важна“ в даден смисъл, и съответно трябва да получи по-висок ранг[7]. Алгоритъмът *TextRank* адаптира тази идея към текстове – те биват моделирани като графи, като според целта текста се моделира по два различни начина – при *извличане на ключови думи*, възлите са последователности от една или повече лексикални единици, а ребрата показват релация на съвместна поява, като свързват единици появяващи се достатъчно близо една до друга в текста. При *извличане на изречения* възлите представят отделните изречения в текста, а ребрата този път са претеглени, и представят сходството на съседни изречения чрез броя общи токени (или други критерии). Това второ приложение е нужното за целите на резюмиране – чрез алгоритъма ще получим ранкиране на изреченията по тяхната „важност“, след което може просто да изберем първите няколко най-важни (според това колко дълго искаме да бъде резюмето) да участват във финалното резюме.

Латентен семантичен анализ (*Latent Semantic Analysis*, *LSA*, също познат и като *Latent Semantic Indexing*)[11] е техника за анализиране на връзките между едно множество документи и термините които те съдържат, чрез създаване на множество от понятия (или теми, *topics*), които ни дават различни начини да интерпретираме значението на документите. За тази цел се създава матрица с броя срещания на всяка дума за всеки документ, и към нея се прилага *singular value decomposition* – алгебрична техника, с която матрицата *M* се декомпозира до три нови, *M = UΣVT*. След няколко операции на умножение на матрици, получаваме *document-topic* таблица, която ни показва за всеки документ доколко „принадлежи“ на всяка тема, претеглено по относителната важност на съответната тема. Така всъщност вече не разглеждаме индивидуалните думи формиращи текстовете, а *темите* които те обсъждат, и получаваме по-компактно представяне[12]. За целите на резюмиране, чрез тези стойности за важност може да определим най-представителните изречения за включване в резюмето.

## Използвани технологии, платформи и библиотеки

За реализиране на проекта са избрани езика *Python* и методите *TextRank* и *LSA*.

Има няколко реализации на *TextRank* на *Python*, като тук е избрана реализацията *PyTextRank*[6], която освен стандартния алгоритъм включва и два други от фамилията на базираните на графи алгоритми – *PositionRank*, който променя ранкирането за да отчете позициите на думите (и така подобрява резултатите), и *Biased TextRank*, който изпълнява случайни рестартирания според важността на възлите в графа спрямо фокуса на задачата (зададен чрез допълнителен списък от токени).

За *LSA* също има няколко реализации, като избраната тук е в библиотеката *sumy*[13], която включва много различни алгоритми и техники за резюмиране освен *LSA*.

За оценка на резултатите се използва множеството от метрики *ROUGE*[8], които измерват сходството между машинното резюме което даден алгоритъм е създал, и друго (обикновено ръчно-направено) резюме, т.нар. *reference*. За нея също има няколко реализации на *Python*, като избраната в проекта е *Rouge* на Paul Tardy[9]. Има няколко *ROUGE* метрики, като тази конкретна реализация включва три от тях – два варианта на *ROUGE-N* (която оценява припокриването на N-грами), със стойности съответно N=1 и N=2, и *ROUGE-L* (която оценява най-дългата обща под-последователност)[10].

Като допълнителни библиотеки използвани в проекта, важно е да се спомене *spaCy* – библиотека за обработка на естествен език, която служи като основа на *PyTextRank*. В *spaCy*, обработката се извършва чрез *pipeline*, като документът последователно минава през няколко *pipe*-а – *PyTextRank* е имплементиран точно като един такъв *pipe*, който се добавя към обект на *spaCy* (заедно с други, вградени в *spaCy* pipe-ове, като *tokenizer*, *POS tagger*, *lemmatizer*, и други).

Други използвани библиотеки са *argparse*, за интерфейса през командния ред, и *pprint* за по-добре форматирано показване на резултатите от Rouge, както и всичко, изисквано от горните библиотеки като dependency.

Относно използването им:

* за *PyTextRank*, първо се инициализира *Language* обект на *spaCy* с даден езиков модел, и към него се добавя pipe за *TextRank* (или някой от другите два алгоритъма включени в библиотеката). След това, за самото резюмиране, обектът се прилага (като функция, чрез *\_\_call\_\_*) към текст (представен като низ), при което текстът се обработва от различните pipe-ове, което включва токенизация, *part-of-speech tagging*, парсване на синтактични зависимости, задаване на атрибути на токените, лематизация, *named entity recognition*, и финално – изпълнението на избрания алгоритъм. Като резултат получаваме *Doc* обект на *spaCy*, от който може да извлечем *n*-те най-важни изречения, и да получим самото резюме като просто ги конкатенираме.
* за *LSA* от *sumy*, първо се създават няколко обекта от класове на *sumy* – *Stemmer* и *Summarizer* (който използва Stemmer-а), и за резюмирането чрез трети обект – *PlaintextParser*, се създава *Document* обект за даден текст, към него се прилага *Summarizer*-а, и подобно получаваме първите *n* най-важни изречения, които след това дообработваме за да получим резюмето като единствен низ.
* за *Rouge*, негова функция получава два еднакво-дълги списъка от резюмета – създадените от алгоритъма, и reference резюметата с които ще ги сравняваме. Според булев аргумент на функцията, може или да получим списък от индивидуални резултати за всяка двойка резюмета, или единствен резултат, осреднен по всички двойки.

## Подготвяне на данните

За целите на проекта бяха намерени множество корпуси от данни, като бяха избрани конкретно три – *DUC 2004*[14], *WikiHow-Dataset*[15] и *Multi-News*[16]. Първият и третият са от новинарски статии, докато вторият е от статии от сайта *wikiHow*, фокусиращ се върху инструктивни how-to статии на най-различна тематика.

Корпусът *DUC 2004* всъщност включва документи за няколко различни задачи, като само първата група беше подходяща за целите на проекта. Организирането на документите е нестандартно – всъщност не са предоставени индивидуални документи, а 50 клъстера (с около 10 документа във всеки), като всеки клъстер е фокусиран върху конкретна новина или тема, а документите в него представят статии за нея от различни източници. Според информацията на сайта на *DUC*, би трябвало за всеки документ да има много кратко резюме, но в dataset-а[14] който аз успях да намеря имаше само резюмета на клъстерите – по 4 на всеки, като те не са обвързани с конкретен документ, а само с общата тема на клъстера. За целите на проекта, просто бяха избрани първия документ и резюме от всеки клъстер, за общо 50 двойки документ-резюме. Не се налагаше допълнителна обработка на данните, те вече са в обикновен текстов формат.

Корпусът *WikiHow* е изключително масивен, покриващ над 230 000 статии. За целите на проекта беше избрано само тяхно подмножество – всяка 500на статия (като в крайна сметка се получиха само 364 двойки документ-резюме, поради документи даващи *encoding* грешки, които просто бяха прескочени). В този случай се наложи допълнителна обработка – dataset-а е в *.csv* формат, така че първо трябваше да бъде прочетен (с помощта на библиотеката *pandas*), да бъдат извършени някои допълнителни обработки върху текста (описани в GitHub страницата[15] на dataset-а), и финално всяка отделна статия да бъде записана в отделен текстов файл (и съответно и всяко отделно резюме). Кодът за това е в *WikiHow-Dataset-construction\process.py*.

Корпусът *Multi-News* по принцип е предназначен за мулти-документно резюмиране, но беше използван тук за обикновено резюмиране на отделни документи, един-по-един. Той също е масивен – над 250 000 статии, така че отново беше избрано тяхно подмножество, този път просто първите 300 статии. Авторите предоставят предварително обработен вариант на данните, така че се наложи само минимална обработка – те са в два текстови файла (съответно за статиите и резюметата), като на всеки ред на файловете е по една статия/резюме. Просто файловете бяха прочетени ред по ред, и беше направено малко допълнително форматиране на получените низове (всяко изречение поставено на нов ред) за да бъдат по-лесно четими от човек. Низовете бяха записани във файлове, по един файл за всяка статия. Кодът за това е в *MultiNews-Dataset-construction\process.py*.

## Реализация

Някои детайли от реализацията са изяснени в раздела *Проектиране*. Имаме следните класове:

* *Corpus*
  + Инициализира се с име на корпус, който да бъде прочетен от диска, и диапазон на документи от него (по подразбиране се чете целия корпус). Запазва документите и резюметата в списъци, и инициализира обект на Rouge, който ще извършва оценяването.
  + Има функция *get\_rouge*, която приема списък от резюмета, сравнява ги със собствените чрез *Rouge* и връща двойка от индивидуалните и осреднените резултати.
  + Функциите *score\_and\_print* и *score\_and\_write* получават списък от резюмета (и име на папка при втората), и формират резултатите във формат на низове, съдържащи за всеки документ - оригиналния текст, полученото резюме, *Rouge* резултатите. Първата функция отпечатва това на конзолата, а втората го записва на диска, в отделен файл за всеки документ.
* *AbstractSummarizer*, и два негови наследника:
* *TextrankSummarizer*
  + Инициализира се по описания в 4.2 начин.
  + Функцията *process\_text* обработва единствен текст, а *process\_corpus* обработва цял корпус (получен като аргумент), по описания в 4.2 начин, и според булев аргумент извиква *score\_and\_print* или *score\_and\_write* с резултатите си.
  + Функцията *get\_summary* се използва при обработката за получаването на самото резюме, а *check\_pipes* е за тестване – тя показва pipe-овете на *spaCy Language* обекта, с което може да се провери че съответния резюмиращ алгоритъм наистина е добавен коректно.
* *LSASummarizer*
  + Инициализира се по описания в 4.2 начин.
  + Функциите *process\_text* и *process\_corpus* работят подобно, но позовавайки се на друг алгоритъм – детайлите на работата са описани в 4.2.

При изпълнение на скрипта, първо с *argparse* се парсват аргументите от командния ред, създава се корпус според подаденото име (и диапазон, ако има такъв), създава се обект от съответния клас според избрания тип резюмиране (и езиков модел в случая на някой от *PyTextRank* алгоритмите), и с този обект се извършва резюмирането. Според аргумента *write*, резултатите или ще бъдат отпечатани на конзолата, или записани на диска, в папката на корпуса, в под-папка указваща името на ползвания алгоритъм (плюс езиковия модел при *PyTextRank*) и броя изречения, до които са били ограничени резюметата.

Относно тестването – за трите корпуса, бяха генерирани резюмета ползвайки *TextRank*, *PositionRank* и *LSA*, като за всеки алгоритъм отделно бяха генерирани резюмета от по 3 изречения, и от по 5 изречения. За всяко резюме се генерира файл със самото него, оригиналния текст за сравнение, *Rouge* резултатите от сравнение с *reference* резюмето, и осреднените *Rouge* резултати за целия корпус. Допълнително бяха генерирани и резюмета на *DUC* с *TextRank* при друг езиков модел - *en\_core\_web\_lg*, който е 57 пъти по-голям (като файлов размер) от стандартния *en\_core\_web\_sm*, използван преди това. При по-големия модел обаче нямаше значително подобрение, а по-големия му размер значително забави изпълнението на програмата, така че не беше използван отново.

По-долу може да се види графика на *F1* оценката на *ROUGE-L* метриката (*ROUGE* метриките имат три оценки – *r (recall)*, *p (precision)*, и *F1*, която се сформира от другите две).

Метриката *ROUGE-1* дава малко по-добри резултати (с ≈0.2-0.3 повече за *DUC*, ≈0.1-0.2 повече за *WikiHow*, но с малко по-доброто ≈0.4-0.5 за *Multi-News*), докато *ROUGE-2* дава значително по-лоши (≈0.045 и надолу за *DUC* и *WikiHow*, ≈0.10-0.12 за *Multi-News*).

Един фактор, допринасящ към по-лошите резултати за *DUC* и *Wiki-How* е свързан с *reference* резюметата.

Както беше описано в 4.3, в *DUC* резюметата не са обвързани с конкретен документ, а с темата на клъстера, тоест с множеството от документи със същата тематика. Така резюметата не са базирани на фрази и детайли от един единствен документ, а на общото знание получено от всичките документи заедно. Тъй като за целите на проекта от всеки клъстер е избран по само един документ, то често се получава че *reference* резюмето звучи по доста различен начин.

Абстрактивен метод потенциално би се справил по-добре, тъй като би генерирал оригинални изречения (подобно на хората писали резюметата, които не са цитирали директно, а са перифразирали информацията), и дори и екстрактивен метод е възможно да получи по-добър резултат при мулти-документно резюмиране (ползвайки целия клъстер вместо само първия документ от него, както всъщност е формулирана втората задача в *DUC 2004*), но тук резюмираме по един документ от темата, и екстрактивната същност на методите означава че получаваме само директни цитати – а в *reference* резюмето най-вероятно няма срещнем същите цитати. Това може би обяснява и значително по-лошите резултати на *ROUGE-2* – когато в *ROUGE-1* гледаме само отделни думи, все пак намираме съвпадения, но когато търсим биграми, различния израз в резюметата намалява срещанията на съвпадащи словосъчетания.

При *WikiHow* има друг проблем с резюметата – те всъщност не са ръчно-написани професионални резюмета, а са взети директно от самите статии, чрез конкатенация на първите редове на всеки параграф (в *wikiHow*, типично всеки параграф започва с едно изречение в удебелен шрифт, което би трябвало да е нещо като мини-резюме на съответния параграф). Това в някои случаи може наистина да дава качествено резюме, но имайки предвид масивния брой статии и изключителното разнообразие в тяхната тематика, вероятно има много случаи в които така получените резюмета не са особено подходящи (например в статия в която се изреждат неща, удебелените изречения може да споменават само името на нещото, без никаква съществена информация за него). Така резюмиращите алгоритми ще избират напълно различни изречения като значещи, при което се получава разминаване с *reference* резюмето и съответно липса на съвпадения, което води до лоши *ROUGE* резултати.

# Заключение

В проекта са приложени основно два екстрактивни алгоритъма за резюмиране – *TextRank* и *LSA*, като допълнително е приложен и *PositionRank* (който е близък по функционалност до *TextRank*). Алгоритмите не са имплементирани ръчно, използвани са *Python* библиотеки с техни реализации.

За тестване са избрани три корпуса от данни, от тях е избрано подмножество от документи, и е извършена допълнителна обработка на данните когато се е налагало. Всеки алгоритъм е приложен за всеки корпус, като резултатите са оценени с *ROUGE* метрики, и не са особено добри във всички случаи, но все пак алгоритмите успешно създават резюмета.

Като бъдещо развитие, възможно е към проекта да бъдат добавени още алгоритми, както и да се усъвършенства работата на текущите. Например, тук размерът на резюмето се определя като входен параметър, еднакъв за всички текстове при едно конкретно изпълнение на програмата (като при тестовете за стойности са избрани 3 и 5 изречения), което може да не е най-подходящия вариант, тъй като различните документи може да имат значителни разлики в размерите им. Ако един документ се състои от 20 изречения, а друг от 50, но и за двата документа създадем резюме от 3 изречения, то най-вероятно ще имаме по-лоши резултати за втория. Динамично определяне на размера на резюмето (като някакъв процент от размера на оригиналния документ) потенциално би подобрило резултатите.

# Използвана литература

1. “Automatic summarization”, *Wikipedia*

[<https://en.wikipedia.org/wiki/Automatic_summarization>]

1. Luís Gonçalves, “Automatic Text Summarization with Machine Learning — An overview”, *Medium - Learning from data, Data Mining and Statistics*, Apr 11, 2020

[<https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25>]

1. Abhijit Roy, “Understanding Automatic Text Summarization-1: Extractive Methods”, *Towards Data Science*, Aug 6, 2020

[<https://towardsdatascience.com/understanding-automatic-text-summarization-1-extractive-methods-8eb512b21ecc>]

1. “Towards Automatic Summarization. Part 2. Abstractive Methods.”, *Sciforce*, Feb 4, 2019

[<https://medium.com/sciforce/towards-automatic-summarization-part-2-abstractive-methods-c424386a65ea>]

1. Rada Mihalcea and Paul Tarau, “TextRank: Bringing Order into Texts”, *EMNLP*, July 1, 2004

[<https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf>]

1. derwen.ai, PyTextRank - a Python implementation of *TextRank*, *GitHub* [<https://github.com/DerwenAI/pytextrank>]
2. Xu Liang, “Understand TextRank for Keyword Extraction by Python”, *Towards Data Science*, Feb 18, 2019

[<https://towardsdatascience.com/textrank-for-keyword-extraction-by-python-c0bae21bcec0>]

1. Chin-Yew Lin, “ROUGE: A Package for Automatic Evaluation of Summaries”, *Association for Computational Linguistics*, 2004

[<https://aclanthology.org/W04-1013/>]

1. Paul Tardy, Rouge - A full Python librarie for the ROUGE metric, *GitHub*

[<https://github.com/pltrdy/rouge>]

1. James Briggs, “The Ultimate Performance Metric in NLP”, *Towards Data Science*, Mar 4, 2021 [<https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460>]
2. “Latent Semantic Analysis”, *Wikipedia*

[<https://en.wikipedia.org/wiki/Latent_semantic_analysis>]

1. Ioana, “Latent Semantic Analysis: intuition, math, implementation”, *Towards Data Science*, May 10, 2020

[<https://towardsdatascience.com/latent-semantic-analysis-intuition-math-implementation-a194aff870f8>]

1. Mišo Belica, Automatic text summarizer, *GitHub*

[<https://github.com/miso-belica/sumy>]

1. DUC 2004: Documents, Tasks, and Measures, *Document Understanding Conferences*, 2004

[<https://duc.nist.gov/duc2004/>]

[<https://github.com/UsmanNiazi/DUC-2004-Dataset>]

1. Mahnaz Koupaee, William Yang Wang, “WikiHow: A Large Scale Text Summarization Dataset”, *University of California, Santa Barbara*, 18 Oct 2018

[<https://arxiv.org/abs/1810.09305>]

[<https://github.com/mahnazkoupaee/WikiHow-Dataset>]

1. Alexander R. Fabbri, et al., “Multi-News: a Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model”, *Association for Computational Linguistics*, 2019

[<https://128.84.21.199/pdf/1906.01749.pdf>]

[<https://github.com/Alex-Fabbri/Multi-News>]