Project task

Summarising a text document

Dmytro Tkachenko

Lenart Marjanovič

Paul Douat

28. 05. 2024

1. **Description of the problem**

We aim to research and to develop a program that automatically generates a shortened summary for an input document by identifying internal sentences which capture key aspects around its subject matter while ignoring less relevant details. This task will involve using Latent Semantic Indexing (LSI) as our method due to its reliance on statistical principles such as word frequency trends among other things in order to achieve better precision levels than would otherwise be possible just through manual selection, which involves several mathematical and computational steps to process the text and extract the most informative sentences.

The examples of the retrieval summarizing sentences will be provided based on first 3 chapters of the novel by George Orwel “*1984*”.

We did 32 tests with different properties: the number of dimensions for matrix factorization, the number of summarizing sentences in the end for manual selection of the best of the bests, 2 different methods to construct the relational matrix between terms and sentences, also the factor of whether the text was preprocessed or not was included. All of them are attached to the project.

1. **Description of the mathematical model**

Latent Semantic Indexing (LSI) is a common technique in Natural Language Processing and information retrieval that uses singular value decomposition (SVD) to identify patterns in the relationships between terms and concepts contained in unstructured text data.

* 1. **Text processing**

Let's imagine a typical sentence in English, which consists of a large number of articles, prepositions, different forms of verbs and other parts of speech. Unfortunately, LSI is not smart and strong enough to ignore them, instead our learning technique will take these words into account and it will recognize verbs like "*go*", "*going*", "*goes*", "*went*" and "*gone*" as different subject words by meaning. And the conjecture generated will be based on them instead of searching for more appropriate words.

Especially for this purpose, the text should always be preprocessed to avoid a negative impact of the stopwords by the following techniques:

1. Stopwords removing. The first part of the text normalization, it requires from us simply removing of anything unrelated to contextualization. Articles, prepositions, conjunctions, interjection, determiners, and even pronouns can be removed.
2. Lemmatization. It is for sure one of the most important methods for preprocessing in NLP (Natural Language Processing) which can be especially helpful when we have a lot of different forms of the same verb, and we don’t want to take into an account separation of verbs in simple, infinitive or third person singular. Lemmatization will reduce words to its lemmas or roots. For example: “*running*” will be easily translated into “*run*”, etc...

The examples of the difference between using the LSI in processed and not processed text is shown a bit later.

**2.1.2 Comparison of the results**

Example without text normalization:

1. *"He could never see the face of Goldstein without a painful mixture of emotions."*
2. *"Its smooth creamy paper, a little yellowed by age, was of a kind that had not been manufactured for at least forty years past."*

1. *"She was a bold-looking girl, of about twenty-seven, with thick hair, a freckled face, and swift, athletic movements."*
2. *"Little boy screaming with fright and hiding his head between her breasts as if he was trying to burrow right into her and the woman putting her arms round him and comforting him although she was blue with fright herself, all the time covering him up as much as possible as if she thought her arms could keep the bullets off him."*
3. *"A colourless, crushed-looking woman, with wispy hair and a lined face, was standing outside."*

There is only 1 good representable sentence and it is the first one as this sentence encapsulates Winston's internal conflict and the Party's manipulation of emotions, central themes not only in the Chapter I, but in *"1984"* in general.

Example with text normalization:

1. "All subsequent crimes against the Party, all treacheries, acts of sabotage, heresies, deviations, sprang directly out of his teaching."
2. *"The thing that now suddenly struck Winston was that his mother's death, nearly thirty years ago, had been tragic and sorrowful in a way that was no longer possible."*
3. *"Someone whom the old man loved--a little granddaughter, perhaps--had been killed."*
4. *"Then she buried her face in her hands."*
5. *'B-B!' always filled him with horror."*

In this collection of sentences, we can easily already get better result: 2 good summarizing sentences: #1 and #2. Where the first one highlights to us the code conflict between individual thought and the Party’s posture, which is crucial to the story. The second one expresses to us the most valuable problem for the society in 1984: lost humanity and emotional devastation.

* 1. **Matrix Construction:**

We constructed a term-sentence matrix 𝐴 using TF-IDF algorithm (term frequency – inverse document frequency), where each row represents a unique term, and each column represents a sentence. The element in the matrix denotes the frequency of the 𝑖-th word in the 𝑗-th sentence.

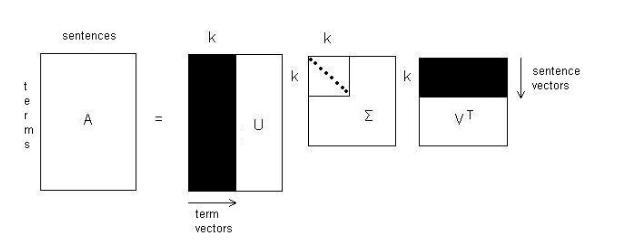


**Figure 1. TF-IDF matrix construction (Harry Potter corpus)**

Sentences selected:

1. *“The broomstick flew higher and higher, and the faster it flew, the more excited Ron became, wishing he could fly even higher*.”
2. *“In the wizarding world, wands choose the wizard, and each wizard's wand is as unique as the wizard who wields it.”*
3. *“Harry, the young wizard, knew that being a wizard meant facing dangers, but Harry also knew that a true wizard never backs down”*
4. *“Harry looked up at Dumbledore, who was now wearing a hat that looked like a turkey.”*
5. *“Harry’s last month with the Dursleys wasn’t fun.”*
6. *“Malfoy always envied Harry as he was a better wizard.”*
   1. **Singular Value Decomposition (SVD):**

After we decomposed the matrix 𝐴 into three matrices U, Σ and such that . Here, U and 𝑉 are orthogonal matrices, and Σ is a diagonal matrix containing singular values. Their roles are very important in following analysis.

****But now we need to take a small break to talk about each of them.

**Figure 1: Applied Singular Value decomposition for our solution**

1. U orthogonal matrix corresponds to terms, each row vector there is a term-vector.
2. orthogonal matrix is responsible for another very important part. It contains sentence-vectors as column vectors.
3. derives a mapping between U and matrices. The larger the value of eigenvalue, the greater the connection between corresponding term and sentence.

We should retain only the largest singular values to reduce dimensionality and capture the most significant relationships in the data, which helps in mitigating overfitting.

* 1. **Sentence Selection**

For each significant singular value in Σ, we identified the corresponding sentence with the largest component. These sentences are deemed most representative of the document's content.

Alternatively, we used also a weighted length measure considering the singular values to select sentences.

* 1. **Optimizations**

The problem is that using a default counting of each term in each sentence is not as efficient as we sometimes want to have and received results can be sufficiently improved and can be made more representable and balanced using different methods to optimize Latent Semantic Indexing.

One of the most important and robust methods involves *term weighting*. To make this step further we need to reconsider our matrix construction. This method takes singular values into account when selecting sentences by weighting the components of by the corresponding singular values.

A weighted score is calculated for each sentence by multiplying the absolute components of the vectors by the corresponding singular values. Using of this method can lead to sufficient improvement of the performance because each weight also considers global weight of each term.

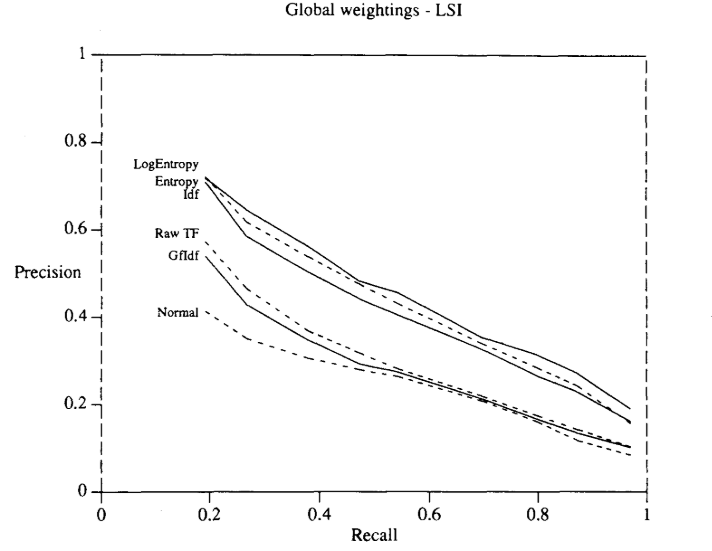
Potentially more accurate because it uses the weighting information to assess the importance of each sentence.

The value will be calculated by the equation where is a local weight of the term i in the sentence j and G(i) is a global weight of the term i overall in the collection of the observed documents. One of the best methods to find local weight is the following formula:

Global weight can be calculated using different methods such as Normal, GfIdf, Idf and the method based on entropy. Even though the entropy method is much more complicated to compute, this report will cover it more because it has nicer properties and better retrieval results:

Here *l* – entropy and *n* – the number of documents is the collection. is a probability of the word in the sentence which is calculated by the following formula:

Where is the global frequency of the term *i* in the collection of documents.

**Figure 3. Global weighting methods comparison**

The experiment was taken on document abstracts in aeronautics and related areas used in Cranfield Institute of Technology, England.

We can easily notice LogEntropy results has the best retrieval precision for the same recall.

* + 1. **Comparison of the results**

Example with global weightings

Was added in **2.1.2** in the example with included text normalization.

Example with TF-IDF method

1. *"But there was a fraction of a second when their eyes met, and for as*

*long as it took to happen Winston knew--yes, he KNEW! -that O'Brien was thinking the same thing as himself."*

1. *"In his waking thoughts he called it the Golden Country."*
2. *"At this moment, however, even the monologue had dried up."*
3. *"But there was a fraction of a second when their eyes met..."*

Only based on the first sentence here we can introduce the idea: how significant the connection between Winston and O’Brien actually is, foreshadowing their interactions and contributing in the future.

1. **Sentence evaluation**

As the method to evaluate the best summarizing sentence was taken content-based similarities method, which computes the similarity at a more fine-grained level than just sentences. We will use this formula point here is to use cosine distance between between the full text of document and its summary.

where *x* and *y* are full document and selected contender to be a summarizing sentence.

1. **Choosing the k for TSVD**

Choosing the number of dimensions for Truncated SVD is an interesting problem, which usually involves into selecting this value by randomly sampling and checking which result is best.

The impact of the number k to the result obtained can be catastrophic. In case of proper selection, it will remove unnecessary noise from the data, otherwise the model will select more vague sentences or on the contrary it will skip more worthy candidates for the role of "The most summarizing sentence".

* 1. **Examples**

Example with 20 singular values:

Was added in **2.1.2** in the example with included text normalization.

Example with 50 singular values:

1. *“Then she buried her face in her hands”*
2. *“Someone whom the old man loved--a little granddaughter, perhaps--had been killed”*
3. *“For a second, two seconds, they had exchanged an equivocal glance, and that was the end of the story”*
4. *“It was, he now realized, because of this other incident that he had suddenly decided to come home and begin the diary today”*
5. *"B-B!' always filled him with horror”*

As the result no meaningful summarizes we can find.

1. **Description of the software code**

The software code to perform the above steps was written in Python, because of its simplicity and comprehensibility to the reader. To our project we also added libraries such as NumPy for numerical computations and Scikit-learn for text processing. Here is an outline of the code:

* 1. The function that is responsible for processing text, dividing it into sentences and building term-sentence matrix using TF-IDF:

def build\_word\_sentence\_matrix(document):

"""

Builds a word-sentence frequency matrix from a document.

Args: document (str): The document to process.

Returns: tuple: A tuple containing the word-sentence matrix, words, and sentences.

  """

    # Split document into sentences and strip whitespace

    sentences = [sentence.strip() for sentence in document.split('.') if sentence.strip()]

    # Initialize the count vectorizer with stop words filtering

vectorizer = CountVectorizer(stop\_words='english')

    X = vectorizer.fit\_transform(sentences)

    # Convert the matrix to an array and get the feature names (words)

matrix = X.toarray()

words = vectorizer.get\_feature\_names\_out()

return matrix, words, sentences

5.2 SVD computing

def truncated\_svd(matrix, energy\_threshold=0.9):

"""

Performs truncated Singular Value Decomposition (SVD) on a matrix.

Args: matrix (numpy.ndarray): The matrix to decompose, energy\_threshold (float): The energy threshold to retain in the decomposition.

Returns: tuple: The truncated U, S, V matrices and the number of singular values used.

"""

# Perform SVD on the matrix

U, S, Vt = svd (matrix, full\_matrices=False)

total\_energy = np.sum(S)

running\_energy = 0.0

k = 0

    # Retain the top singular values based on the energy threshold

    while running\_energy / total\_energy < energy\_threshold and k < len(S):

        running\_energy += S[k]

        k += 1

    U\_k = U[:, :k]

    S\_k = np.diag(S[:k])

    V\_k = Vt[:k, :]

    return U\_k, S\_k, V\_k, k

* 1. Local measure calculating

def calculate\_local\_measure(matrix):

    """

    Calculates the local measure for the word-sentence matrix.

    Args: matrix (numpy.ndarray): The word-sentence frequency matrix.

    Returns: numpy.ndarray: The local measure matrix.

    """

    # Calculate the local measure using the logarithm of the frequency + 1

return np.log(matrix + 1)

* 1. Global measure calculating

def calculate\_global\_measure(matrix):

 """

Calculates the global measure for the word-sentence matrix using entropy.

    Args: matrix (numpy.ndarray): The word-sentence frequency matrix.

Returns: numpy.ndarray: The global measure for each word.

 """

 # Calculate total word frequency for each sentence

 total\_word\_freq = np.sum(matrix, axis=1)

 n = matrix.shape[1]

 global\_measure = np.zeros\_like(total\_word\_freq, dtype=float)

 # Calculate the global measure based on entropy for each word

 for i in range(matrix.shape[0]):

     if total\_word\_freq[i] > 0:

        p\_ij = matrix[i] / total\_word\_freq[i]

        entropy = -np.sum(p\_ij \* np.log(p\_ij + 1e-10)) / np.log(n)

         global\_measure[i] = 1 - entropy

     else:

        global\_measure[i] = 1

return global\_measure

* 1. Build complex matrix from the equation

def build\_complex\_matrix(matrix, local\_measure, global\_measure):

    """

    Builds a complex matrix by combining local and global measures.

    Args: matrix (numpy.ndarray): The word-sentence frequency matrix,

local\_measure (numpy.ndarray): The local measure matrix,

global\_measure (numpy.ndarray): The global measure for each word.

    Returns: numpy.ndarray: The complex matrix.

    """

    complex\_matrix = np.zeros\_like(matrix, dtype=float)

    # Combine local and global measures to form the complex matrix

    for i in range(matrix.shape[0]):

        for j in range(matrix.shape[1]):

            complex\_matrix[i, j] = local\_measure[i, j] \* global\_measure[i]

return complex\_matrix

1. **Conclusion**

Provided research and the code above have shown to us that using LSI we can generate a summary of the given document very precisely, especially if we take into account pros and cons of different methods and use optimization. Also, the relation of different number of Singular values in TSVD, the presence of text preprocessing on the result. The summary consists of sentences that capture the main topics and information presented in the document, ensuring that the essence of the content is retained in a concise form.