University of Piraeus

Natural Language Processing

Loukas Kokotos P22249 Stavros Kolouas P18077 Fani Zerva P22046

Contents

Introduction

In the world we live in, where we need fast solutions and we communicate with many people from different linguistical backgrounds, services like **translation**, quick **summaries** and **virtual assistance** are not only expected, but needed. These are all made possible with **Natural Language Processing** (NLP). It is a field that bridges **human language** and **computational logic and understanding** in order to provide and extend certain services. Its importance is **significant**, as it is the reason machines can assist us humans and understand what we need. Such services are chatbots, translation and even protection from unwanted emails (spam). However, we will use it to do some simpler tasks: **reconstructing texts and sentences to achieve better tone and grammar**.

Deliverable 1

Task A

Goal

The purpose of the assignment is to reconstruct two sentences from two texts in order to achieve better grammar and tone. The sentences of choice are: “Hope you too, to enjoy it as my deepest wishes” and “Because I didn’t see that part final yet, or maybe I missed, I apologize if so.”. It is clear that they are problematic, poorly structured and unclear when out of context, therefore in need of fixing. In order to achieve that, our program will cover the rearrangement of the words in the sentence to fit the norm, as well as the addition of missing words and deletion of unnecessary ones. For the sake of the reformed sentences to be the closest possible to the original ones, we treated abbreviations like “didn’t” as two words split before and after the apostrophe. This will not affect the process greatly, but needs to be clarified here for it to appease speculations. The definition of words is the first order of business for proper **tokenization**, as the paper on **Speech and Language Processing** from Standford indicates.

Procedure

Tokenization

First of all, we need to start with the implementation of the basics of NLP. We have to break down the sentence to each word so we can examine and analyze it. This process is called “**tokenization**”. It is the first crucial step and it is partially responsible for the program to be deterministic. In order

to achieve our goal, we use two libraries. **Re** (regular expression) helps indicate the boundaries of each word and **nltk** (Natural Language Toolkit) contains useful heuristics to create our tokens.

Parsing

Since we have the tokens of the sentences, we now need to create a parser to analyze the grammatical meaning of each word in the sentences. First, we have to categorize each word in terms of the part of speech it belongs to. For example, the word “wishes” is a noun and therefore will go into N, which is the corresponding label. In the same section we have to write down the **grammatic rules** the sentences follow by breaking them down, since we are implementing **Context-Free Grammar** (CFG). After grouping them up, we set the sequence the sentences follow by including all the different verbal and noun phrases they consist of in a series of “or” statements. After the grammar is complete, the two parse trees are created from the given tokens and are showcased in the terminal.

Reconstructing

The next step in the procedure is to take the parsed tokens and apply grammatic rules to them. For example, a rule is for the adverb to be the last word in a sentence. However, as we have clarified above, our goal is not just to reposition the words, but add missing ones and keep the ones necessary. For that purpose, we add missing nouns, like “I” in the begging of the first sentence and remove the “to”. These modifications significantly improve the tone, preserve the meaning and tackle ambiguity.

Task B

Creating a custom program for the purpose of reconstructing two specific sentences was a challenging, but manageable task. The same cannot be said for reconstructing the two whole texts. To achieve that goal, we will be using some python **libraries** and **pipelines**. We will examine three different approaches to see what different variations will be generated, which will later on be compared.

Components

We will use the library **re** once more for **pattern recognition** and matching between words. We will also use **sklearn** in order to turn text into a representing **numerical vector** and to calculate the **similarity** between the generated text and the original. Lastly, **numpy** will contribute to finding the maximum **similarity score**.

Method 1

The first approach is the simplest one, since it is very custom and meant to be applied to the specific texts. It also lacks scalability, which affects its atomization. It is a rule-based pipeline, that improves the badly written sentences, by replacing them with some predefined ones. Whenever the algorithm goes through a phrase that fits the discretion given, it will change it with the one that we consider an improvement. As a result, we get two texts with better grammar and clear meaning.

Method 2

Our second approach is more advanced, since it is **more scalable**. We use a list of words under the variable "**candidates**", which contains words that paraphrase our text. The candidates are then made into numerical vectors in order to be compared with one another. The phrases are compared based on **cosine similarity**, a very effective and smart technique that we have practiced in class, as its focus is on **preserving the meaning**. After that, the most similar phrase is picked and replaced in the text, leading to a better written text with **improved tone**.

Method 3

The final approach is fixated only in improving the vocabulary and therefore tone of the texts. Many of the words are problematic, as they are either very informal, fail to represent the ideas of the author and confuse the reader. A solution for this problem consists of presenting cleverly picked synonyms. The words flagged as "problematic" are being replaced with their efficient synonyms. However, this solution fixes only one of our two problems. Confusion, clearance and ambiguity are addressed, but grammar structure stays the same.

Task C

In this section we will look a little closer on the techniques we used above.

Task A

As we already mentioned, we used Context – Free Grammar (CFG) in order to analyze and parse the sentences. With NLTK the given text is split into words that are made into tokens. The tokens get rearranged to fit proper syntax, while some get deleted or added. The strongpoint of this method is that it can spot grammatical errors, is **tailor-made** for the sentences given and is deterministic. Its downside is the **limitations** it has. Because the grammar is completely custom and specific for each sentence, it cannot handle different texts without major changes.

Task B

Rule-based pipeline

The Regex (regular expression) rules, applied with this method, replaced **specific phrases** in order to have a clear output. The process was easy and **fast**, having predefined patterns and available substitutes. However, it once again lacks flexibility exactly because it relies on **anticipated phrases** and **patterns**.

TF-IDF and cosine similarity

Our second approach creates **multiple versions** of each sentence and picks the one closest to the original, based on the TF-IDF model and cosine similarity of each variation. While it's more flexible due to the lack of strict rules, its **pure statistical criteria** do not allow it to "understand" the sentences. That leads to the selection of the replacement words to be the best mathematically, but not always the best linguistically. Furthermore, the syntactical errors are not being fixed, as it does not analyze structure.

Synonym replacement pipeline

The third and final approach reforms the texts using **synonyms** to improve the quality of the words and the tone of the text. While this is a fast and can handle more **variety** than the first pipeline, it lacks understanding of context. This makes it **inefficient**, as it can trigger ambiguity, confusion and transform the tone into an inappropriate one.

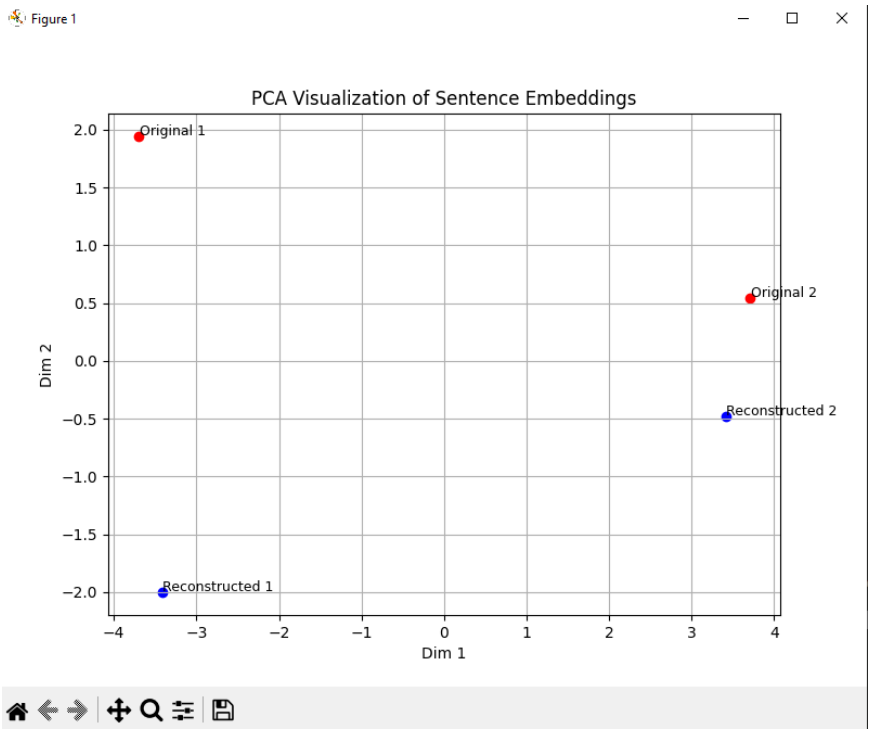
Deliverable 2

Firstly, we load the original and reconstructed sentences of task 1A. Even though we are aware that the rephrasing was correct and preserved the meaning, in this task we will examine that theory with the help of **visualized graphs**. The next step in this procedure is to parse the sentences with word embeddings. We chose **BERT embeddings**, as they rely on context and are high dimensional. BERT analyses the sentences and creates its tokens by understanding subwords and assigns them an ID. With this tool the differences will be clearer and better examined, as each of the words is also represented with a high dimensional **vector**.

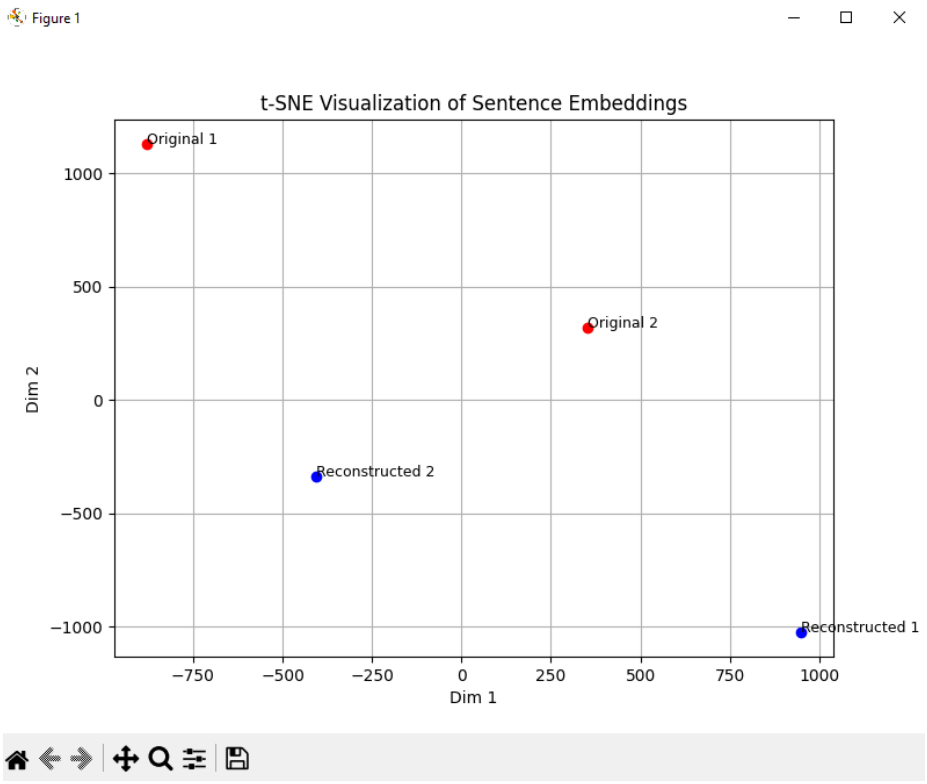
After the tokens are created, BERT takes each of their vectors and creates an **average value** that represents each sentence as a whole. These vectors are then compared via **cosine similarity**. Meanwhile, the four sentences -two original and two reconstructed- are being assigned points in a **two-dimensional matrix**, in order to see the similarities and differences between them. For this purpose, we will use both **PCA** (Principal Component Analysis) and **t-SNE** (t-distributed Stochastic Neighbor Embedding). This decision was made because each technique represents a different approach and we wanted to see what we truly achieved. PCA focuses on the **structure** and is **deterministic**, for accurate results, while t-SNE examines mostly **local similarity**.

Results

Looking at the first graph, which used PCA, we see that the sentences share similarities in pairs. While the first pair -original 1 and reconstructed 1- are almost **perfectly** in sync on the one dimension, they **differ significantly** in the other one. On the other hand, the other pair appears



to be very close on both dimensions. This result may seem weird, especially since the **cosine similarity score** of the two pairs was **very high** (>0.9) and they were both generated with the **same technique**. However, the distances make a lot of sense considering the **PCA ideology**. Like we mentioned before, it mostly compares the structure. The first sentence was written worse then the second one, leading to bigger transitions between the two. Therefore, this result is valid and since they are similar on the other dimension, it's considered a successful reconstruction.



On the second graph we immediately notice that the pairs' distances are bigger. However, this decline is also **justifiable** considering the multiple words that were switched and the t-SNE's focus on **local similarity**. Since these switches lead to a better tone and the cosine similarity score is bigger than 0.9, this graph just indicates our **improvements**.

Final thoughts

Word embeddings managed to translate the words into vectors for the machine to understand and compare rather effectively. That was made clear mostly in the previous section since we had a visual representation to go along with the cosine similarity. The results were the ones expected, therefore this tool is deemed both **helpful** and **accurate**. Without it we would not be able to visualize our reconstructions and truly understand the outputs.

The whole process had many challenges. Thankfully, we had great material, which helped understand the philosophy, steps and goals. The biggest one had to be the first. As with most things, the biggest roadblock is to start. However, it was also the most important thing. By just using re and nltk for context free grammar we had to **really think about the process** and **understand the basic** ideas of nlp. After we got that down, the use of libraries and pipelines became **easier** and more **intuitive**. It was the **hardest** step, but by far the most **important** one.

For learning purposes, a completely automated process would not be ideal, as students could rely heavily on pre made libraries and tools and never grasp the meaning and logic of NLP. However,

such a thing **can be made**. By using **more libraries** for better **paraphrasing** (Pegasus), **grammar correction** (Gramformer) and the tools we used and described, rewriting sentences with improved tone, vocabulary and structure is possible. It is safe to assume that such a program would outperform ours, but that is only natural. Afterall, the goal of this assignment was to learn NLP, not to make the perfect constructor.

Finaly, there were noticeable differences in the quality of reconstruction among the different techniques, methods, and libraries used. The **rule-based automaton** produced very **precise** and predictable outputs, but was **limited** to predefined phrases. The **TF-IDF with cosine similarity** was more **flexible** and required fewer manual rules, but its statistical nature meant that it could not truly understand meaning or syntax, resulting in outputs that were sometimes close to the original text but **not necessarily grammatically correct**. The **synonym replacement approach** improved **lexical variety** by substituting words with synonyms, but it often **ignored context**, which sometimes led to awkward or inconsistent sentences. Finally, the **CFG Parser with NLTK** delivered the **most correct structured results**, as it relied on grammar rules to rearrange and correct sentences. However, it was also the most complex to implement and worked only on relatively simple sentence structures.

Bibliography

Speech and Language Processing, 3rd Ed. — Jurafsky & Martin

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Devlin et al., 2019

https://github.com/dimitris1pana/nlp_lab_unipi

Class material