NLP task for I.I. Mechnikova Programming Course

Research project done
by **George Kokkin**

PROBLEM SPECIFICATION

For the given text, count how many times the transition from one word to another occurred within sentences, and print the first 100 most common transitions [1]. Build a directed graph in SVG that shows transitions of words within a sentence. At each transition, indicate how many times the transition from the first word to the second occurred in the text. Generate 100 random sentences in the style of the selected text. Sentences must begin with a capitalised word and end with a period or an exclamation or question mark. Optionally, make sure that the frequency of generated phrases corresponds to their frequency in the text.

INTRODUCTION

To solve a given problem, Python programming language will be used, as it has multiple libraries and convenient interfaces for data manipulations. A class

“TextGenrator” which holds all the functions for working with a given text will be implemented. All the information related to the usage of a tool, running script and getting dynamic results is stored into README.md file.

SOLUTION STEPS

Data retrieving and cleaning

Having a URL with a target text, Python library Beautiful Soap is used to extract text part of the entire HTML page [2]. Right after, a substitution is applied using a regular expression pattern to delete all the dashes and commas from the text.

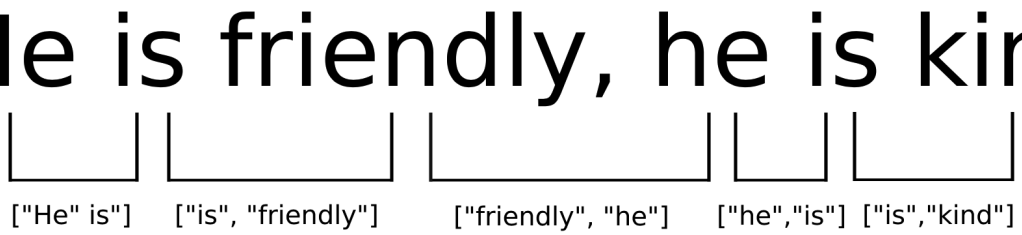
```
def __get_text_from_url(cls, url_: str) -> str:
    extracted_text: str = ''
    html_doc: str = requests.get(url_).text
    soup = BeautifulSoup(html_doc, 'html.parser')
    for p in soup.find("h1", string="The Signal-Man").find_parent('div').find_all('p'):
        for br in p.find_all('br'):
            br.replace_with(" ")
        if "breadcrumb" in p.__str__():
            break
        extracted_text += p.text
    return extracted_text
```

```
def clean_text(text):
    return re.sub(r'-', ' ', re.sub(r'[",]', '', text))
```

Calculating transitions

To calculate all the transitions means to get the frequency of every sequential word pair from the text (e.g., “he is” - 1 occurrence, “He is” - 1 occurrence). Such pairs as a last with the first word of a next sentence are not considered. To get

desired data, a helper function for splitting text by sentences and splitting sentence by words were implemented using regular expressions. In the example below, there are 5 different pairs, each has a frequency of 1.



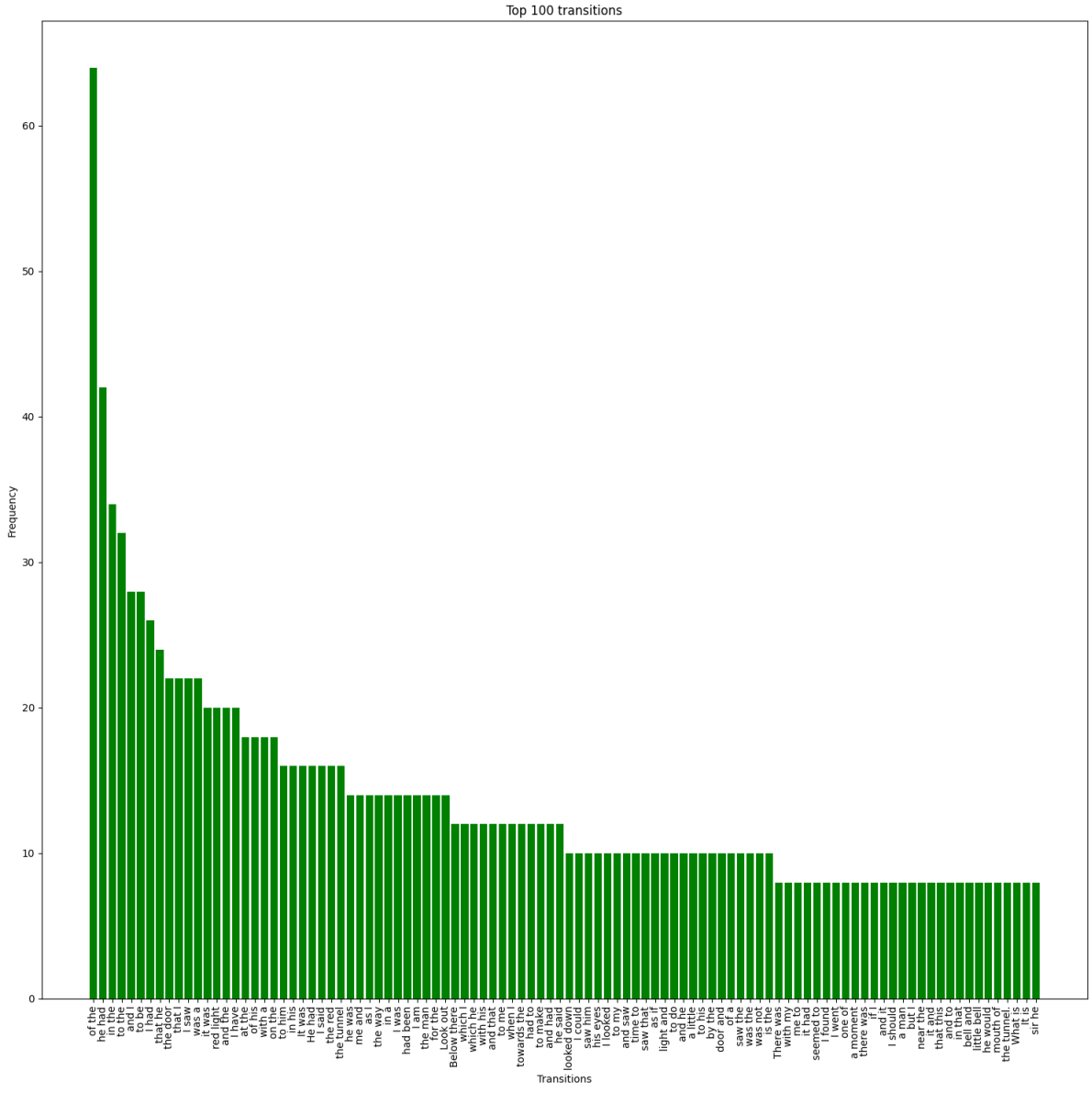
Within a function for calculating transitions, the outer loop picks every sentence, while the inner loop goes through every pair sequentially, adding 1 to the frequency of the particular pair. Frequency data itself is a dictionary (map), where an array of two words is a key, and its frequency is a value (e.g. map(key = [“he”, “is”], value = 1.0)). A *defaultdict* data type is used to automatically fill nonexistent before keys with a default value, in our case is 0.

```
def split_text_by_sentence(text):
    return re.split(r'(?<=[.!?])\s*', text)
```

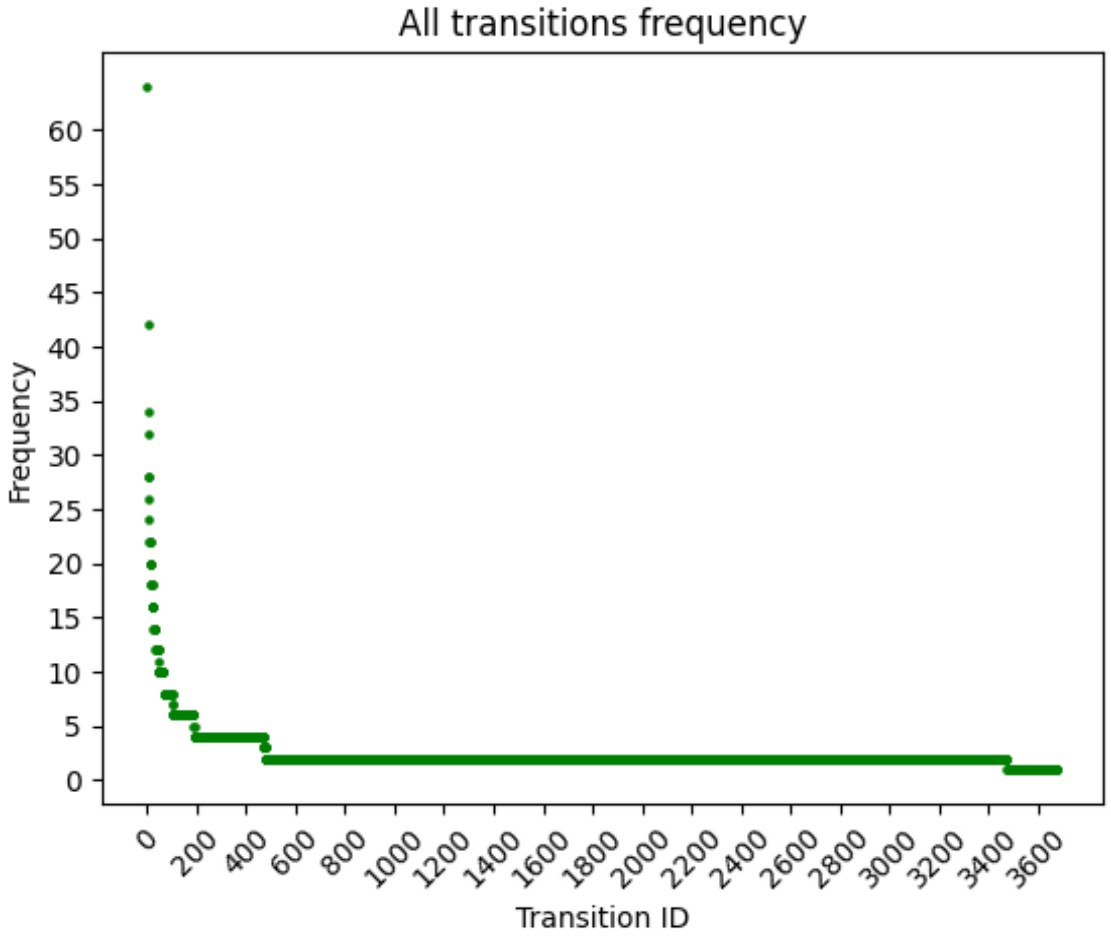
```
def split_sentence_by_words(sentence):
    return re.findall(r'\b\S+\b\.*', sentence)
```

```
def __calculate_transitions(cls, text: str):
    transitions: defaultdict = defaultdict(float)
    for sentence in split_text_by_sentence(text):
        words: list = split_sentence_by_words(sentence)
        for i in range(len(words) - 1):
            transitions[(words[i], words[i + 1])] += 1
    return transitions
```

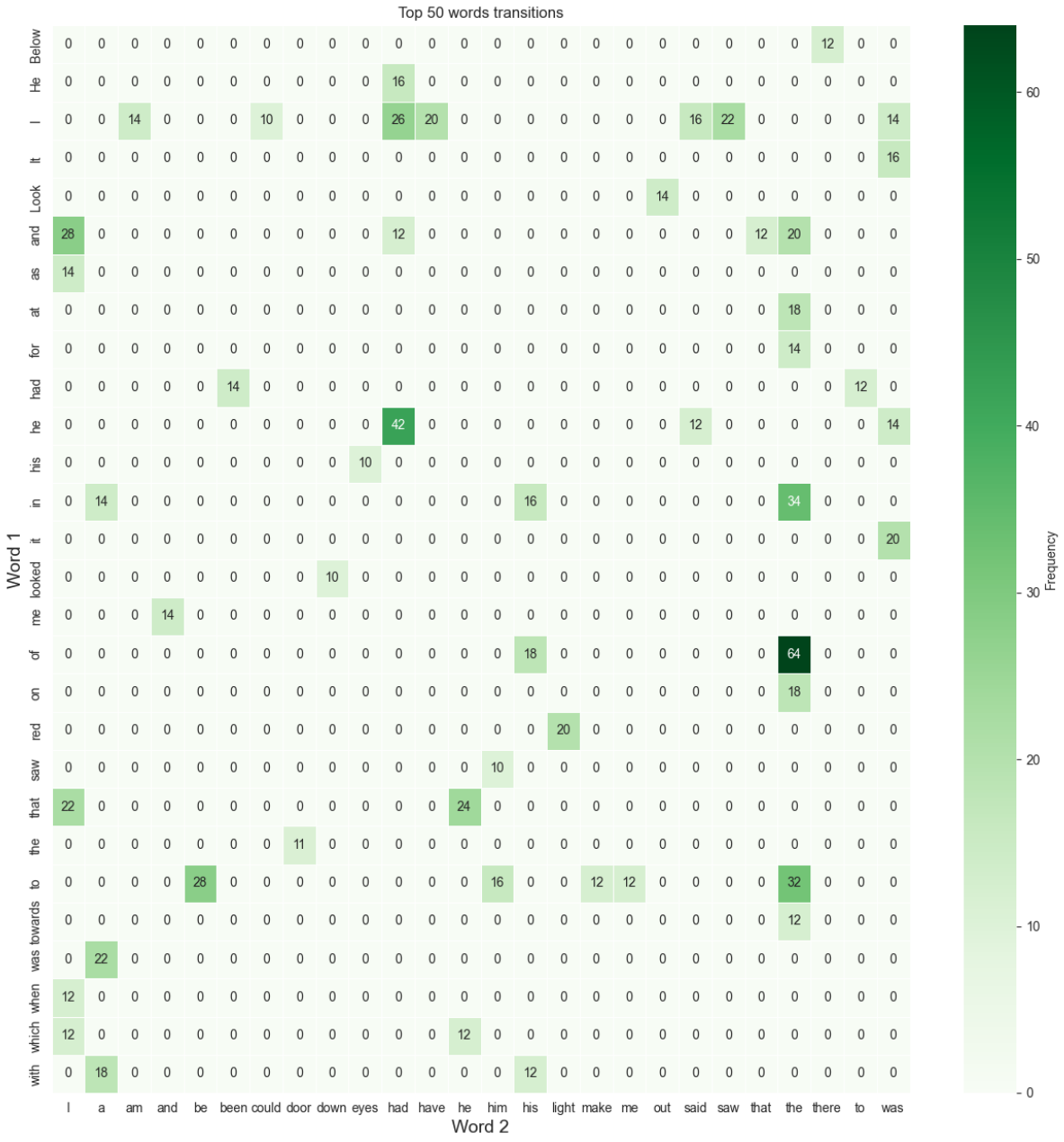
Transitions data was visualised as a graph using MatPlotLib library [3]. On a figure below, which shows top 100 word transitions, it can be noticed that almost all of them are meaningless - linking words, pronouns and auxiliary verbs, excluding exceptions. With a significant gap of ~20 from the second most popular, a pair “of the” is the most common word sequence in a text, that occurred more than 60 times, “he had” - 42 times, which could indicate that a main character of the story is a male.



On the next figure, the frequency of all 3678 pairs is graphed. It could be clearly seen that before ~500th sequence frequency is pretty high, while it is falling firstly rapidly, then gradually. Heading towards the right corner, remaining data has a value of 2, which is a mode. The average frequency is ~2.503.

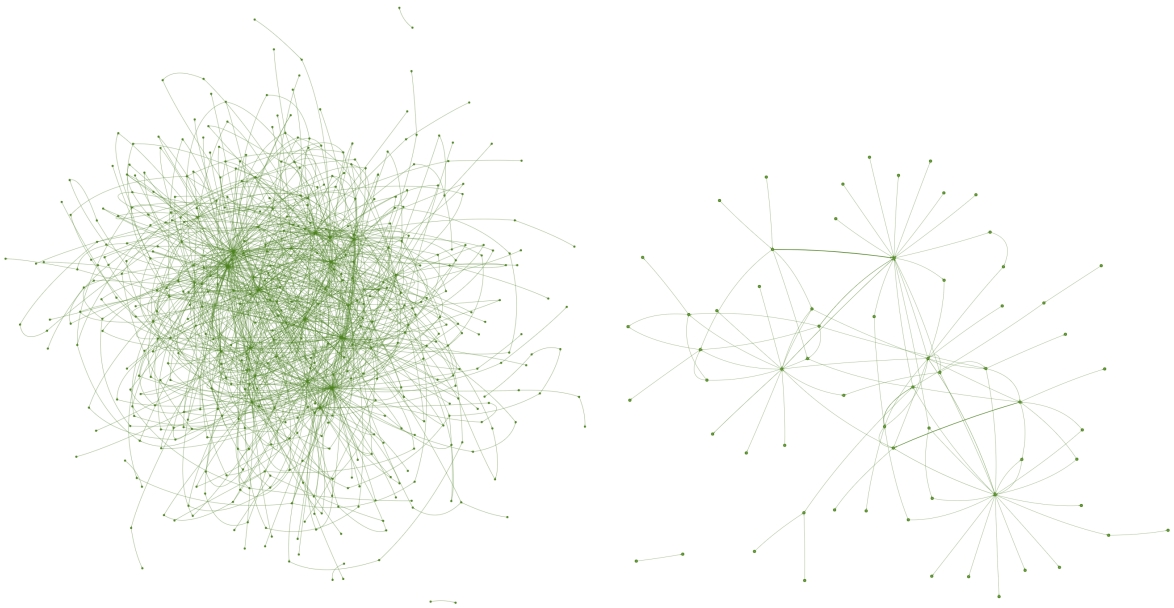


Another approach to show the results is heatmap. Here, we are able to see the top 50 most common transitions and their frequency within an intersection of a pair, which is indicated both by colour saturation and numerical value into the cell.



In order to build a directional graph, the pyvis library is used [4]. It outputs an HTML document with an interactive graph. A code shows a method for setting it up. Iterating our transitions data, each time we add two nodes as a first and a second word, and then connect two passed graph edges with a line, weight of which is set by pair frequency. The sample output graphs are shown below.

```
for key in data_to_vis:
    graph.add_node(n_id=key[0], label=key[0], title=key[0], color="#6da33e")
    graph.add_node(n_id=key[1], label=key[1], title=key[1], color="#6da33e")
    graph.add_edge(key[0], key[1], value=data_to_vis[key])
```



Generating data

The next step is a new (pseudo) data generation. In the snippet below which generates 1 sentence, following steps are present (working with transitions data gained before):

1. Pick up a random capitalised word, which is a first word of some and our sentence
2. Find such pairs where the previously picked word (step 1) would be the first word of a sequence.
3. From all the found variants randomly pick the one, while the higher frequency leads to higher chance of being chosen. In order to keep the same pairs frequency as in original text, a frequency would be decremented by 1 each time a pair used.
4. If the found word is terminating (includes punctuation mark), finish the sentence, if not - repeat the process.

```
def generate_sentence(self) -> str:
    George Kokkin *
    def get_next_word(curr_word: str):
        next_word_choices = [(transition[1], weight) for transition, weight in self.transitions_data.items() if
                             transition[0] == curr_word]
        if len(next_word_choices) == 0:
            return False
        words, weights_ = zip(*next_word_choices)
        choice: str = random.choices(population=words, weights=weights_)[0]
        self.transitions_data[(curr_word, choice)] -= 1
        return choice

    sentence: list = []

    entry_word: str = random.choice(
        list(set([transition[0] for transition in self.transitions_data if
                  transition[0][0].isupper()])))

    while True:
        sentence.append(entry_word)
        if entry_word[len(entry_word) - 1] in '.!?':
            break
        entry_word = get_next_word(entry_word)
        if not entry_word:
            break

    return " ".join(sentence)
```

An example of 2 generated sentences could be seen below. Notice that the sentence is that long because it terminates only if a word with a punctuation mark is picked.

Sir from what was a language down the tunnel. Nothing came out of other I heard a dreadful time except when I saw the level of mind is the way long winter nights there would rarely be rehearsing the Danger on both hands and dismal mouth of the word Sir from mine was a matter of his grave dark regards divided between us that something of his sense that I have set this arm with a tone but what he slowly added these reasons I did this rapid train had directed them no doubt because I saw the figure had looked it may have slept but it off messages and then was above him said I.

CONCLUSION

In today's work a text was splitted into small pieces, processed and analysed. Based on a 2 words sequence frequency, pseudo new sentences were generated. As an answer to the given problem, a primitive class-based tool was built with Python3 and its libraries.

SOURCES

1. Given text. URL: https://www.online-literature.com/dickens/2941/
2. Beautiful Soap. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
3. Data visualisation library (barchart, heatmap). URL: https://matplotlib.org/
4. Graph visualisation library. URL: https://pyvis.readthedocs.io/