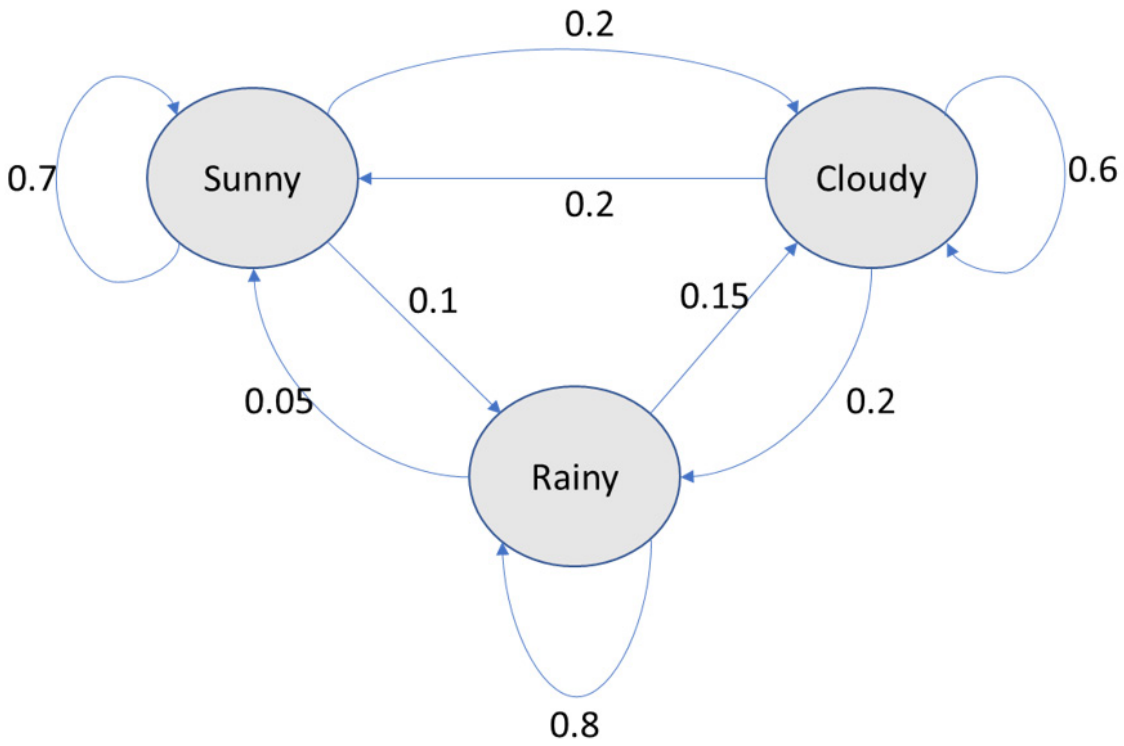
Lab 7. Text Generation and Summarization

Introduction

The ability to express thoughts in words (sentence generation), the ability to replace a piece of text with different but equivalent text (paraphrasing), and the ability to find the most important parts of a piece of text (summarization) are all key elements of using language. Although sentence generation, paraphrasing, and summarization are challenging tasks in NLP, there have been great strides recently that have made them considerably more accessible. In this lab, we explore them in detail and see how we can implement them in Python.

Generating Text with Markov Chains

A state space defines all possible states that can exist. A Markov chain consists of a state space and a specific type of successor function. For example, in the case of the simplified state space to describe the weather, the states could be Sunny, Cloudy, or Rainy. The successor function describes how a system in its current state can move to a different state or even continue in the same state. To better understand this, consider the following diagram:



Exercise 7.01: Text Generation Using a Random Walk over a Markov Chain

In this exercise, we will generate text with the help of Markov chains. We will use Robert Frost's collection of poems, *North of Boston*, available from Project Gutenberg, to specify the successor state(s) for each state using a dictionary. We'll use a list to specify the successor state(s) for any state so that the number of times a successor state occurs in that list is directly proportional to the probability of transitioning to that successor state.

Then, we will generate 10 phrases with three words in addition to an initial word, and then generate another 10 phrases with four words in addition to an initial word. The initial state or initial word will be randomly selected from among these words: "the," "a," "I," "he," "she," "if," "this," "why," and "where."

Follow these steps to complete this exercise:

1. Open a Jupyter notebook.
2. Insert a new cell and add the following code to import the necessary libraries and read the dataset:

```
```
import re
import random
OPEN_DATA_URL = '../data/robertfrost/pg3026.txt'
f=open(OPEN_DATA_URL,'r',encoding='utf-8')
text=f.read()
f.close()
```
```

3. Insert a new cell and add the following code to preprocess the text using regular expressions:

```
```
HANDLE = '@\w+\n'
LINK = 'https?://t\.co/\w+'
SPECIAL_CHARS = '<|<|&|#'
PARA='\n+'
def clean(text):
    #text = re.sub(HANDLE, ' ', text)
    text = re.sub(LINK, ' ', text)
    text = re.sub(SPECIAL_CHARS, ' ', text)
    text = re.sub(PARA, '\n', text)
    return text
text = clean(text)
```
```

4. Split the corpus into a list of words. Show the number of words in the corpus:

```
corpus=text.split()
corpus_length=len(corpus)
corpus_length
```

The preceding code generates the following output:

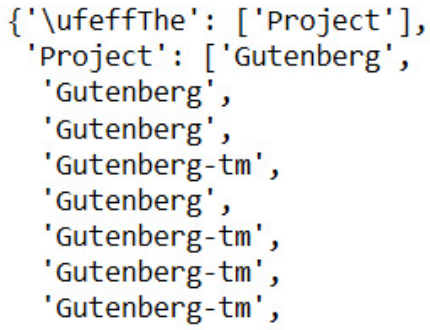
```
19985
```

5. Insert a new cell and add the following code to define the successor states for each state. Use a dictionary for this:

```
succ_func={}
corpus_counter=0
for token in corpus:
    corpus_counter=corpus_counter+1
    if corpus_counter<corpus_length:
```

```
        if token not in succ_func.keys():
            succ_func[token]=[corpus[corpus_counter]]
        else:
            succ_func[token].append(corpus[corpus_counter])
succ_func
```

The preceding code generates an output as follows. **Note** that we're only displaying a part of the output here.




```
We find that \"he\" is shown as a successor of \"who\" more than
once. This is because this occurs more than once in the dataset. In
effect, the number of times the successors occur in the list is
proportional to their respective probabilities. Though it is not the
only method, this is a convenient way to represent the successor
function.
```

6. Define the list of initial states. Then, define a function to select a random initial state from these and concatenate it with successor states. These successor states are randomly selected from the list containing successor states for a specific current state. Add the following code to do this:

```
```
initial_states=['The','A','I','He','She','If',\
                'This','Why','Where']
def generate_words(k=5):
    initial_state=random.choice(initial_states)
    current_state=initial_state
    text=current_state+' '
    for i in range(k):
        succ_state=random.choice(succ_func[current_state])
        text=text+succ_state+' '
        current_state=succ_state
    print(text.split('.')[0])
```
```

7. Insert a new cell and add the following code to generate text containing 10 phrases of four words (including the initial word) and 10 phrases of five words (including the initial word):

```
for k in range(3,5):
    for j in range(10):
        generate_words(k)
```

The preceding code generates the following output:

```
A horse's hoof pawed
She let me go!"
This and hear some
If you must include
A man's affairs
The rumbling voice said
The lawyer said
The Purple Lady's Slipper
Why didn't see at
If he's kinder than
She does she couldn't lead
He has gone
Where you couldn't keep in
This is your efforts and
The Seven Caves that was
She reached the neighbours, Being
If you 'AS-IS' WITH NO
Why not so With white-faced
I said there and any
Why didn't think we know
```

Text Summarization

Automated text summarization is the process of using NLP tools to produce concise versions of text that preserve the key information present in the original content. Good summaries can communicate the content with less text by retaining the key information while filtering out other information and noise (or useless text, if any). A shorter text may often take less time to read, and thus summarization facilitates more efficient use of time.

- Abstractive summarization as a combination of understanding the meaning and expressing it in fewer sentences. It is usually considered as a supervised learning problem as the original text and the summary are both required.
- Extractive summarization: Parts of the text are extracted to form a summary. There is no paraphrasing in this form of summarization.

TextRank

TextRank is a graph-based algorithm (developed by Rada Mihalcea and Paul Tarau) used to find the key sentences in a piece of text. As we already know, in graph theory, a graph has nodes and edges. In the TextRank algorithm, we estimate the importance of each sentence and create a summary with the sentences that have the highest importance.

The TextRank algorithm works as follows:

1. Represent a unit of text (say, a sentence) as a node.
2. Each node is given an arbitrary importance score.

3. Each edge has a weight that corresponds to the similarity between two nodes (for instance, the sentences `Sx` and `Sy`). The weight could be the number of common words (say, *w*[k]{.subscript}) in the two sentences divided by the sum of the number of words in the two sentences. This can be represented as follows:

$$Similarity(S_x, S_y) = \frac{|(W_k : W_k \in Sx \ \& \ W_k \in Sy)|}{log|Sx| + log|Sy|}$$

4. For each node, we compute a new importance score, which is a function of the importance score of the neighboring nodes and the edge weights (*w*[ji]{.subscript}) between them. Specifically, the function (*f*) could be the edge-weighted average score of all the neighboring nodes that are directed toward that node that is adjusted by all the outward edge weights (*w*[jk]{.subscript}) and the damping factor (*d*). This can be represented as follows:

$$f(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{W_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} f(V_j)$$

5. We repeat the preceding step until the importance score varies by less than a pre-defined tolerance level in two consecutive iterations.
6. Sort the nodes in decreasing order of the importance scores.
7. The top *n* nodes give us a summary.

Key Input Parameters for TextRank

We'll be using the gensim library to implement TextRank. The following are the parameters required for this:

- `text`: This is the input text.
- `ratio`: This is the required ratio of the number of sentences in the summary to the number of sentences in the input text.

Exercise 7.02: Performing Summarization Using TextRank

In this exercise, we will use the classic short story, *After Twenty Years* by O. Henry, which is available on Project Gutenberg, and the first section of the Wikipedia article on Oscar Wilde. We will summarize each text separately so that we have 20% of the sentences in the original text and then have 25% of the sentences in the original text using the gensim implementation of the TextRank algorithm. In all, we shall extract and print four summaries.

In addition to these libraries, you will need to import the following:

```
from gensim.summarization import summarize
summarize(text,ratio=0.20)
```

In the preceding code snippet, `ratio=0.20` means that 20% of the sentences from the original text will be used to create the summary.

Complete the following steps to implement this exercise:

1. Open a Jupyter notebook.
2. Insert a new cell and add the following code to import the necessary libraries and extract the required text from *After Twenty Years*:

```
from gensim.summarization import summarize
import wikipedia
import re
file_url_after_twenty=r'../data/ohenry/pg2776.txt'
with open(file_url_after_twenty, 'r') as f:
        contents = f.read()
start_string='AFTER TWENTY YEARS\n\n\n'
end_string='\n\n\n\n\n\nLOST ON DRESS PARADE'
text_after_twenty=contents[contents.find(start_string):\
                           contents.find(end_string)]
text_after_twenty=text_after_twenty.replace('\n',' ')
text_after_twenty=re.sub(r"\s+"," ",text_after_twenty)
text_after_twenty
```

   The preceding code generates the following output:

```
'AFTER TWENTY YEARS The policeman on the beat moved up the avenue impressively. The impressiveness was hab
itual and not for show, for spectators were few. The time was barely 10 o\'clock at night, but chilly gust
s of wind with a taste of rain in them had well nigh de-peopled the streets. Trying doors as he went, twir
ling his club with many intricate and artful movements, turning now and then to cast his watchful eye adow
n the pacific thoroughfare, the officer, with his stalwart form and slight swagger, made a fine picture of
a guardian of the peace. The vicinity was one that kept early hours. Now and then you might see the lights
```

3. Add the following code to extract the required text and print the summarized text, with the `ratio` parameter set to `0.2`:

```
summary_text_after_twenty=summarize(text_after_twenty, \
                                    ratio=0.2)
print(summary_text_after_twenty)
```

   The preceding code generates the following output:

```
Now and then you might see the lights of a cigar store or of an all-night lunch counter; but the majority
of the doors belonged to business places that had long since been closed.
About that long ago there used to be a restaurant where this store stands--'Big Joe' Brady's restaurant."
"Until five years ago," said the policeman.
"Twenty years ago to-night," said the man, "I dined here at 'Big Joe' Brady's with Jimmy Wells, my best ch
um, and the finest chap in the world.
```

4. Insert a new cell and add the following code to summarize the text and print the summarized text, with the `ratio` parameter set to `0.25`:

```
summary_text_after_twenty=summarize(text_after_twenty, \
                                    ratio=0.25)
print(summary_text_after_twenty)
```

The preceding code generates the following output:

```
Now and then you might see the lights of a cigar store or of an all-night lunch counter; but the majority
of the doors belonged to business places that had long since been closed.
About that long ago there used to be a restaurant where this store stands--'Big Joe' Brady's restaurant."
"Until five years ago," said the policeman.
"Twenty years ago to-night," said the man, "I dined here at 'Big Joe' Brady's with Jimmy Wells, my best ch
um, and the finest chap in the world.
```

5. Insert a new cell and add the following code to extract the required text from the Wikipedia page for Oscar Wilde:

```
#text_wiki_oscarwilde=wikipedia.summary("Oscar Wilde")
file_url_wiki_oscarwilde=r'../data/oscarwilde/'\
                         'ow_wikipedia_sum.txt'
with open(file_url_wiki_oscarwilde, 'r', \
          encoding='latin-1') as f:
        text_wiki_oscarwilde = f.read()
text_wiki_oscarwilde=text_wiki_oscarwilde.replace('\n',' ')
text_wiki_oscarwilde=re.sub(r"\s+"," ",text_wiki_oscarwilde)
text_wiki_oscarwilde
```

The preceding code generates the following output:

```
'Oscar Fingal O\'Flahertie Wills Wilde (16 October 1854 – 30 November 1900) was an Irish poet and playwrig
ht. After writing in different forms throughout the 1880s, the early 1890s saw him become one of the most
popular playwrights in London. He is best remembered for his epigrams and plays, his novel The Picture of
Dorian Gray, and the circumstances of his criminal conviction for "gross indecency", imprisonment, and ear
ly death at age 46. Wilde\'s parents were successful Anglo-Irish intellectuals in Dublin. A young Wilde le
arned to speak fluent French and German. At university, Wilde read Greats; he demonstrated himself to be a
n exceptional classicist, first at Trinity College Dublin, then at Oxford. He became associated with the e
```

6. Insert a new cell and add the following code to summarize the text and print the summarized text using `ratio=0.2`:

```
summary_wiki_oscarwilde=summarize(text_wiki_oscarwilde, \
                                  ratio=0.2)
print(summary_wiki_oscarwilde)
```

The preceding code generates the following output:

```
He is best remembered for his epigrams and plays, his novel The Picture of Dorian Gray, and the circumstan
ces of his criminal conviction for "gross indecency", imprisonment, and early death at age 46.
As a spokesman for aestheticism, he tried his hand at various literary activities: he published a book of
poems, lectured in the United States and Canada on the new "English Renaissance in Art" and interior decor
ation, and then returned to London where he worked prolifically as a journalist.
Unperturbed, Wilde produced four society comedies in the early 1890s, which made him one of the most succe
```

7. Add the following code to summarize the text and print the summarized text using `ratio=0.25`:

```
summary_wiki_oscarwilde=summarize(text_wiki_oscarwilde, \
                                  ratio=0.25)
print(summary_wiki_oscarwilde)
```

The preceding code generates the following output:

```
He is best remembered for his epigrams and plays, his novel The Picture of Dorian Gray, and the circumstan
ces of his criminal conviction for "gross indecency", imprisonment, and early death at age 46.
As a spokesman for aestheticism, he tried his hand at various literary activities: he published a book of
poems, lectured in the United States and Canada on the new "English Renaissance in Art" and interior decor
ation, and then returned to London where he worked prolifically as a journalist.
Unperturbed, Wilde produced four society comedies in the early 1890s, which made him one of the most succe
```

Note: We find that the summary for the Wikipedia article is much more coherent than the short story. We can also see that the summary with a `ratio` of `0.20` is a subset of a summary with a `ratio` of `0.25`. Would extractive summarization work better for a children's fairytale than it does for an O. Henry short story? Let's explore this in the next exercise.

Exercise 7.03: Summarizing a Children's Fairy Tale Using TextRank

In this exercise, we consider the fairy tale *Little Red Riding Hood* in two variations for the input texts. The first variation is from *Children's Hour with Red Riding Hood and Other Stories*, edited by Watty Piper, while the second variation is from *The Fairy Tales of Charles Perrault*, both of which are available on Project Gutenberg's website. The aim of this exercise is to explore how TextRank (gensim) performs on this summarization.

Complete the following steps to implement this exercise:

1. Open a Jupyter notebook.
2. Insert a new cell and add the following code to import the required libraries:

```
```
from gensim.summarization import summarize
import re
```
```

3. Insert a new cell and add the following code to fetch Watty Piper's version of *Little Red Riding Hood*:

```
file_url_grimms=r'../data/littleredrh/pg11592.txt'
with open(file_url_grimms, 'r') as f:
        contents_grimms = f.read()
start_string_grimms='LITTLE RED RIDING HOOD\n\n\n'
end_string_grimms='\n\n\n\n\nTHE GOOSE-GIRL'
text_grimms=contents_grimms[contents_grimms.find(\
                            start_string_grimms):\
                            contents_grimms.find(\
                            end_string_grimms)]
text_grimms=text_grimms.replace('\n',' ')
text_grimms=re.sub(r"\s+"," ",text_grimms)
text_grimms
```

The preceding code generates the following output:

```
'LITTLE RED RIDING HOOD There was once a sweet little maid who lived with her father and mother in a prett
y little cottage at the edge of the village. At the further end of the wood was another pretty cottage and
in it lived her grandmother. Everybody loved this little girl, her grandmother perhaps loved her most of a
ll and gave her a great many pretty things. Once she gave her a red cloak with a hood which she always wor
e, so people called her Little Red Riding Hood. One morning Little Red Riding Hood\'s mother said, "Put on
your things and go to see your grandmother. She has been ill; take along this basket for her. I have put i
n it eggs, butter and cake, and other dainties." It was a bright and sunny morning. Red Riding Hood was so
```

Riding Hood

4. Insert a new cell, add the following code, and fetch the Perrault fairy tale version of *Little Red Riding Hood*:

```
file_url_perrault=r'../data/littleredrh/pg29021.txt'
with open(file_url_perrault, 'r') as f:
        contents_perrault = f.read()
start_string_perrault='Little Red Riding-Hood\n\n'
end_string_perrault='\n\n_The Moral_'
text_perrault=contents_perrault[contents_perrault.find(\
                                start_string_perrault):\
                                contents_perrault.find(\
                                end_string_perrault)]
text_perrault=text_perrault.replace('\n',' ')
text_perrault=re.sub(r"\s+"," ",text_perrault)
text_perrault
```

The preceding code generates the following output:

```
'Little Red Riding-Hood Once upon a time, there lived in a certain village, a little country girl, the pre
ttiest creature was ever seen. Her mother was excessively fond of her; and her grand-mother doated on her
much more. This good woman got made for her a little red riding-hood; which became the girl so extremely w
ell, that every body called her Little Red Riding-Hood. One day, her mother, having made some girdle-cake
s, said to her: "Go, my dear, and see how thy grand-mamma does, for I hear she has been very ill, carry he
r a girdle-cake, and this little pot of butter." Little Red Riding-Hood set out immediately to go to her g
rand-mother, who lived in another village. As she was going thro\' the wood, she met with Gaffer Wolf, who
```

Hood

5. Insert a new cell and add the following code to generate the two summaries with a `ratio` of `0.20`:

```
```
llrh_grimms_textrank=summarize(text_grimms,ratio=0.20)
llrh_perrault_textrank=summarize(text_perrault,ratio=0.20)
```
```

6. Insert a new cell and add the following code to print the TextRank summary (`ratio` of `0.20`) of Grimm's version of *Little Red Riding Hood*:

```
print(llrh_grimms_textrank)
```

The preceding code generates the following output:

```
LITTLE RED RIDING HOOD There was once a sweet little maid who lived with her father and mother in a pretty
little cottage at the edge of the village.
One morning Little Red Riding Hood's mother said, "Put on your things and go to see your grandmother.
"What have you in that basket, Little Red Riding Hood?" "Eggs and butter and cake, Mr. Wolf." "Where are y
ou going with them, Little Red Riding Hood?" "I am going to my grandmother, who is ill, Mr. Wolf." "Where
```

variation

7. Insert a new cell and add the following code to print the TextRank summary (`ratio` of `0.20`) of Perrault's version of *Little Red Riding Hood*:

```
print(llrh_perrault_textrank)
```

The preceding code generates the following output:

```
"Who's there?" "Your grand-child, Little Red Riding-Hood," replied the Wolf, counterfeiting her voice, "wh
o has brought you a girdle-cake, and a little pot of butter, sent you by mamma." The good grand-mother, wh
o was in bed, because she found herself somewhat ill, cry'd out: "Pull the peg, and the bolt will fall." T
he Wolf pull'd the peg, and the door opened, and then presently he fell upon the good woman, and ate her u
```

version

8. Add the following code to generate two summaries with a `ratio` of `0.5` :

```
```
llrh_grimms_textrank=summarize(text_grimms,ratio=0.5)
llrh_perrault_textrank=summarize(text_perrault,ratio=0.5)
```
```

9. Add the following code to print a TextRank summary (`ratio` of `0.5`) of Piper's version of *Little Red Riding Hood*:

```
print(llrh_grimms_textrank)
```

The preceding code generates the following output:

```
LITTLE RED RIDING HOOD There was once a sweet little maid who lived with her father and mother in a pretty
little cottage at the edge of the village.
At the further end of the wood was another pretty cottage and in it lived her grandmother.
One morning Little Red Riding Hood's mother said, "Put on your things and go to see your grandmother.
Little Red Riding Hood wandered from her path and was stooping to pick a flower when from behind her a gru
ff voice said, "Good morning, Little Red Riding Hood." Little Red Riding Hood turned around and saw a grea
```

10. Add the following code to print a TextRank summary (`ratio` of `0.5`) of Perrault's version of *Little Red Riding Hood*:

```
print(llrh_perrault_textrank)
```

The preceding code generates the following output:

```
One day, her mother, having made some girdle-cakes, said to her: "Go, my dear, and see how thy grand-mamma
does, for I hear she has been very ill, carry her a girdle-cake, and this little pot of butter." Little Re
d Riding-Hood set out immediately to go to her grand-mother, who lived in another village.
The poor child, who did not know that it was dangerous to stay and hear a Wolf talk, said to him: "I am go
ing to see my grand-mamma, and carry her a girdle-cake, and a little pot of butter, from my mamma." "Does
```

Activity 7.01: Summarizing Complaints in the Consumer Financial Protection Bureau Dataset

The Consumer Financial Protection Bureau publishes consumer complaints made against organizations in the financial sector. This original dataset is available at https://www.consumerfinance.gov/data-research/consumer-complaints/#download-the-data. To complete this activity, you will summarize a few complaints using TextRank.

Note

To complete the activity, you will need the `.csv` file in the `data` folder for this lab in your localdirectory.

Follow these steps to implement this activity:

1. Import the summarization libraries and instantiate the summarization model.
2. Load the dataset from a `.csv` file into a pandas DataFrame. Drop all columns other than `Product`, `Sub-product`, `Issue`, `Sub-issue`, and `Consumer complaint narrative`.
3. Select 12 complaints corresponding to the rows `242830`, `1086741`, `536367`, `957355`, `975181`, `483530`, `950006`, `865088`, `681842`, `536367`, `132345`, and `285894` from the 300,000 odd complaints with a narrative. **Note** that since the dataset is an evolving dataset, the use of a version that's different from the one in the `data` folder could give different results because the input texts could be different.
4. Add a column with the TextRank summary. Each element of this column corresponds to a summary, using TextRank, of the complaint narrative in the corresponding column. Use a `ratio` of `0.20`. Also, use a `try-except` clause since the gensim implementation of the TextRank algorithm throws exceptions with summaries that have very few sentences.
5. Show the DataFrame. You should get an output similar to the following figure:

	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	TextRank Summary
132345	Mortgage	FHA mortgage	Loan servicing, payments, escrow account	NaN	I have a trial period of loan modification HAR...	They will not acknowledge my modification, and...
242830	Debt collection	Medical	Cont'd attempts collect debt not owed	Debt is not mine	XXXX XX/XX/XXXX XXXX XXXX, XXXX As a 100 % rat...	XXXX Care coordinator XXXX XXXX contacted the ...
285894	Prepaid card	General purpose card	Managing, opening, or closing account	NaN	I purchased a prepaid credit card from XXXX wh...	Was refused and told I have to use the card.
483530	Debt collection	I do not know	Cont'd attempts collect debt not owed	Debt is not mine	This company has called me XXXX times about a ...	

Summary

In this lab, we learned about text generation using Markov chains and extractive summarization using the TextRank algorithm. We also explored both the power and limitations of various advanced approaches. In the next lab, we will learn about sentiment analysis.