



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektroniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Praca dyplomowa

Generacja tekstu w języku naturalnym na przykładzie TweetBota
Natural language text generation on the example of a TweetBot

Autor:
Kierunek studiów:
Opiekun pracy:

Bartłomiej Tomasz Gawęda
Automatyka i Robotyka
dr hab. inż. Jerzy Baranowski prof. AGH

Kraków, 2022

Spis treści

1	Wstęp.....	4
1.1	Wprowadzenie.....	4
1.2	Cel pracy	4
2	Wprowadzenie do przetwarzania języka naturalnego	5
2.1	Zarys historyczny.....	5
2.2	Przyjęte podejście	5
3	Implementacja	7
3.1	Wykorzystane narzędzia	7
3.2	Pozyskanie bazy danych.....	7
3.3	Tworzenie modelu	8
3.3.1	Wstępne przetwarzanie tekstu	9
3.3.2	Enkoder.....	10
3.3.3	Dekoder	10
3.3.4	Trening.....	11
3.4	Tworzenie pozostałej części aplikacji	12
3.5	Dokumentacja.....	14
3.6	Testowanie.....	15
4	Podsumowanie	16
4.1	Wnioski	16
4.2	Potencjalne kierunki rozwoju.....	16
	Bibliografia	17

1 Wstęp

1.1 Wprowadzenie

Przetwarzanie tekstu w języku naturalnym (ang. natural language processing, NLP) nie jest nowym zagadnieniem, jednak główny obszar rozwoju tej technologii to język angielski. Jest on stosunkowo prostym gramatycznie językiem, co znacząco ułatwia programom komputerowym przetwarzanie i jego „naukę”. Z kolei język polski jest znacznie trudniejszym, oraz mniej używanym językiem, a co za tym idzie istnieje dużo mniej narzędzi umożliwiających przetwarzanie tekstu napisanego po polsku.

1.2 Cel pracy

Celem pracy było stworzenie jednej z wspomnianych w poprzednim akapicie aplikacji w postaci TweetBota, którego zadanie polega na automatycznym generowaniu wiadomości tekstowych (tweetów) w języku naturalnym, a następnie publikowanie ich w serwisie społecznościowym Twitter. Aby rozbudować funkcjonalność programu przyjęto, że treść postów będzie tworzona na podstawie opisów gier i będzie mieć formę streszczenia. Zatem głównym modulem aplikacji jest algorytm dokonujący sumaryzacji tekstu w języku polskim. Przyjęto takie założenia ze względu na fakt iż praca [1] z roku 2021 w języku polskim wyróżnia jedynie 9 narzędzi do sumaryzacji, z czego jedynie NLPer oparty był na mechanizmie abstrakcji.

2 Wprowadzenie do przetwarzania języka naturalnego

2.1 *Zarys historyczny*

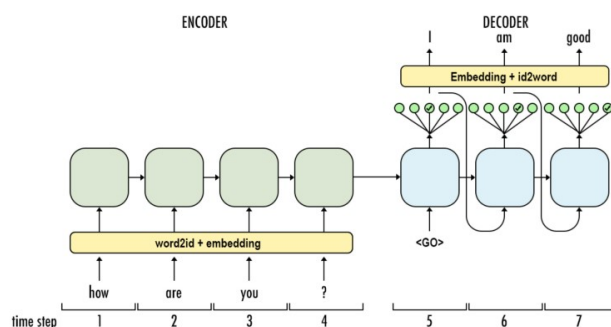
Początki technologii sięgają lat 50 XIX wieku. Jako pierwszego badacza który zajął się tym tematem można podać Alana Turinga, który opublikował artykuł „Machine and Intelligence” traktujący o tym jak zweryfikować czy rozmawiamy z maszyną, czy z człowiekiem [2]. W późniejszych latach wprowadzano różne podejścia do przetwarzania tekstu, jednak na potrzeby pracy skupiono się na sposobach sumaryzacji tekstu. Można wyróżnić trzy uogólnione podejścia podane od tych które pojawiły się najwcześniej, po najnowsze:

- sumaryzację opartą na określonych zasadach – tworzone zbiór zasad opartych na konkretnych frazach, decydujących o tym które fragmenty tekstu najlepiej go streszczają
- sumaryzację opartą na statystyce – tworzona była statystyka słów występujących w tekście i na jej podstawie wybierane były odpowiednie fragmenty do podsumowania całości
- sumaryzację opartą na sieciach – wraz z rozwojem szeroko pojętego uczenia maszynowego, również ta dziedzina skorzystała. Modele tworzone w tej technologii uczone są na wielu przykładach jak powinno wyglądać streszczenie konkretnych tekstów, co pozwala im na sumaryzację innych, nieanalizowanych dotąd tekstów. Można dodatkowo to podejście podzielić dalej na sumaryzację ekstrakcyjną i abstrakcyjną. Pierwsza działa wybierając z tekstu zdania według modelu o największym znaczeniu dla tekstu, druga tworzy nowe zdania na podstawie treści całego analizowanego tekstu.

2.2 *Przyjęte podejście*

W związku z wymaganą charakterystyką generowanych wiadomości (wiadomości umieszczane na Twitterze mają długość ograniczoną do 260 znaków) przyjęto podejście sumaryzacji abstrakcyjnej, gdyż pozwala ono na streszczenie danej wiadomości w mniejszej liczbie słów niż przy podejściu ekstrakcyjnym, ponieważ nie korzysta ze zdań występujących w opracowywanej wiadomości, a na podstawie wyuczonego „rozumienia” tekstu tworzy nowe zdania, w związku z czym jest w stanie zadany tekst znacznie bardziej skompresować.

Do stworzenia modelu wykorzystano architekturę Seq2seq (sequence to sequence) opartą o komórki LSTM



Rysunek 2.1 Schematyczne przedstawienie wybranej architektury [3]

Architektura ta składa się z enkodera i dekodera. Rolą enkodera jest zakodowanie tekstu podlegającego sumaryzacji w postaci $x = (x_1, x_2, \dots, x_J)$ do pewnego stanu ukrytego $h^e = (h_1^e, h_2^e, \dots, h_J^e)$. Następnie dekodery na podstawie tego stanu generuje wiadomość streszczoną w postaci $y = (y_1, y_2, \dots, y_T)$. Gdzie J i T to odpowiednio maksymalne długości tekstu i jego streszczenia podane w ilości słów [4].

3 Implementacja

3.1 Wykorzystane narzędzia

Językiem programowania wykorzystanym podczas realizacji pracy był język Python 3. Aplikacja została stworzona w Dashu [5] – jest to biblioteka umożliwiająca intuicyjne tworzenie aplikacji internetowych – a następnie opublikowana na Heroku [6] – chmurowej platformie umożliwiającej hostowanie aplikacji. Do stworzenia i wytrenowania modelu odpowiedzialnego za generowanie streszczonego tekstu wykorzystano następujące biblioteki:

- Tensorflow [7]
- Keras [8]
- NumPy [9]

W związku ze znacznym zapotrzebowaniem procesu trenowania modelu na zasoby, podczas uczenia modelu wykorzystano internetowe środowisko uruchomieniowe Google Colaboratory [10], pozwalające wykonywać obliczenia w chmurze na wydajniejszym GPU i CPU.

Moduł odpowiedzialny za pozyskanie zbioru danych w głównej mierze został oparty na bibliotece BeautifulSoup [11], Selenium [12] i requests [13]

Pliki tworzone w trakcie projektu umieszczane były w repozytorium w serwisie GitHub [14], co pozwalało na śledzenie pojawiających się błędów, oraz testowanie wgrywanych fragmentów kodu poprzez zastosowanie mechanizmu ciągłej integracji

3.2 Pozyskanie bazy danych

Z względu na fakt, że dla języka polskiego NLP jest stosunkowo niszowym zagadnieniem, nie istnieje zbyt wiele gotowych baz danych pozwalających na przejście do pracy nad modelem, z pominięciem etapu pozyskiwania przykładów do nauki. Dla tematyki tej pracy nie zostały znalezione odpowiednie zbiory, w związku z czym zaszła konieczność stworzenia ich od podstaw. Odpowiednie teksty pozwalające na nauczanie modelu streszczania znaleziono na portalu *www.gry-online.pl*. W serwisie tym znajduje się ranking gier, który zawiera krótki tekst, stworzony na podstawie pełnego opisu produkcji.



Wiedźmin 3: Krew i wino

9.8 Ocena gry

Drugie fabularne rozszerzenie DLC do wydanego w maju 2015 roku RPG akcji Wiedźmin 3: Dzikie Gon. Akcja dodatku zatytułowanego Krew i wino przenosi nas do całkiem nowej krainy Touissant, będącej jednym z niewielu państw wiedźmińskiego uniwersum, którego nie dotknęła wojenna pożoga. Poproszony o pomoc przez księżną Annę Henriettę, wiedźmin Geralt będzie musiał stawić czoła grasującemu w krainie mordercy.

Wiedźmin 3: Krew i Wino (w wersji eksportowej *The Witcher 3: Blood and Wine*) to **drugi duży dodatek DLC do wydanego wiosną 2015 roku rodzimego RPG akcji Wiedźmin 3: Dzikie Gon**. Rozszerzenie wydłużające fabułę pierwowzoru o ok. 30 godzin rozgrywki, opracowane zostało ponownie przez studio CD Projekt RED, czyli twórców wszystkich dotychczasowych odsłon komputerowej serii, bazujące na motywach twórczości znanego polskiego pisarza fantasy Andrzeja Sapkowskiego.

Dodatek *Krew i Wino* został oficjalnie zapowiedziany przez twórców już na miesiąc przed planowaną premierą gry podstawowej i **wraz z**



Rysunek 3.1 Przykładowy krótki opis wraz z fragmentem długiego, na stronie gry-online.pl [15]

Celem pozyskania danych, została stworzona klasa WebScrapper korzystająca z funkcji biblioteki Selenium i BeautifulSoup4. Metody utworzonej klasy pozwalają na nawigację po stronach rankingu, oraz na pobieranie krótkich, jak i pełnych opisów gier. Rezultaty były umieszczane w obiekcie typu DataFrame, a następnie zapisane w formacie CSV. W ten sposób udało się pozyskać 3982 wpisy.

	Game_name	html	Short_text	Opis
0	Wiedźmin 3: Krew i wino	/gry/the-witcher-3-blood-and-wine/zb41ae	Drugie fabularne rozszerzenie DLC do wydanego ...	\nWiedźmin 3: Krew i Wino (w wersji eksportowe...
1	Realpolitiks	/gry/realpolitiks/z847e8	Rozgrywana w czasie rzeczywistym gra strategic...	\nRealpolitiks to rozgrywana w czasie rzeczyw...
2	Deep Diving Simulator	/gry/deep-diving-simulator/zb5769	Polska gra symulacyjna, w której wcielamy się ...	\nDeep Diving Simulator jest grą symulacyjną, ...
3	Flashout 2	/gry/flashout-2/ze3c4d	Kontynuacja futurystycznej gry wyścigowej stud...	\nSequel futurystycznej gry wyścigowej Flashou...
4	Wiedźmin 3: Dzikie Gon	/gry/the-witcher-3-wild-hunt/z83579	Gra action RPG, stanowiąca trzecią część przyg...	\nWiedźmin 3: Dzikie Gon (The Witcher 3: Wild H...
...
3978	Smash up Derby	/gry/smash-up-derby/zdb7d	Dynamiczne wyścigi samochodowe, nawiązujące do...	\nSmash up Derby to dynamiczne wyścigi samocho...
3979	Space Rangers	/gry/space-rangers/zbb54	Połączenie klasycznego kosmicznego symulatora ...	\nPłączenie klasycznego kosmicznego symulator...
3980	The Elder Scrolls III: Trójca	/gry/the-elder-scrolls-iii-tribunal/zeb66	The Elder Scrolls III: Tribunal jest oficjalny...	\nOficjalny, płatny dodatek do The Elder Scrol...
3981	Close Combat	/gry/close-combat/z0af9	Wielokrotnie nagradzana gra strategiczna czasu...	\nPierwsza ze sławnej już serii gier strategic...
3982	Pacific Fighters	/gry/pacific-fighters/z9b34	Pacific Fighters to kolejny po znakomitym IL-2...	\nPacific Fighters to kolejny po znakomitym IL...

Rysunek 3.2 Frgment pozyskanego zbioru danych. Opracowanie własne

3.3 Tworzenie modelu

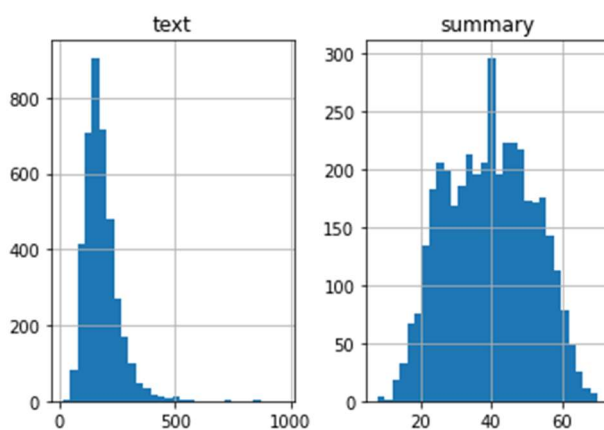
Podczas tworzenia modelu oparto się na rozwiązaniu podobnego problemu umieszczonego na stronie internetowej [16]. Proces doboru ilości warstw przebiegał iteracyjnie – dodawano kolejne warstwy, przeprowadzano proces uczenia modelu i na podstawie otrzymanych wyników modyfikowano model tak, aby otrzymać jak najlepsze rezultaty.

3.3.1 Wstępne przetwarzanie tekstu

Aby tekst mógł być przeczytany przez model, musi on zostać odpowiednio zmodyfikowany. Zaimplementowane w pracy wstępne przetwarzanie tekstu składa się z następujących kroków:

- Wszystkie litery zmieniane są na małe
- Usuwane są wszystkie znaki nie będące literami
- Usuwane są słowa znajdujące się na liście tzw. stop-words. [17] Zawiera ona wyrazy w języku polskim nie niosące ze sobą żadnych istotnych treści (np. a, aby, ach, acz, aczkolwiek, aj...)[18]
- Usuwane są słowa o długości jednej litery, jeśli takie zostały po poprzednich krokach

Następnie teksty są analizowane pod kontem ich długości, dobierana jest maksymalna dopuszczalna długość i teksty które przekraczają ustalony limit są usuwane ze zbioru danych. Ma to na celu ograniczenie wielkości modelu, gdyż nieliczne dłuższe teksty wymagałyby warstwy wejściowej o większym rozmiarze, który to rozmiar wykorzystany byłby jedynie w nielicznych przypadkach.



Rysunek 3.3 Statystyka długości tekstów oraz streszczeń. Opracowanie własne

Dla ograniczenia długości tekstów do 300 słów pokrytych zostało 94%, a dla ograniczenia długości streszczeń do 60 słów pokryto 96% zbioru danych. Finalnym etapem przygotowania danych było stworzenie tokenizera, którego zadaniem polegało na wygenerowaniu słownika pozwalającego zamienić zdania na ciągi liczb naturalnych.

3.3.2 Encoder

Encoder został zbudowany z warstwy embedding oraz z 4 warstw LSTM

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 300)]	0
embedding (Embedding)	(None, 300, 200)	3963200
lstm (LSTM)	[(None, 300, 200), (None, 200), (None, 200)]	320800
lstm_1 (LSTM)	[(None, 300, 200), (None, 200), (None, 200)]	320800
lstm_2 (LSTM)	[(None, 300, 200), (None, 200), (None, 200)]	320800
lstm_3 (LSTM)	[(None, 300, 200), (None, 200), (None, 200)]	320800
=====		
Total params: 5,246,400		
Trainable params: 5,246,400		
Non-trainable params: 0		

Rysunek 3.4 Podsumowanie utworzonego enkodera. Opracowanie własne

3.3.3 Dekoder

Na dekodek składa się warstwa embedding oraz 2 warstw LSTM. Dodano również warstwy wejściowe pozwalające dekodekowi analizować stan wygenerowany przez enkoder.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, None)]	0	[]
embedding_1 (Embedding)	(None, None, 200)	668200	['input_2[0][0]']
input_6 (InputLayer)	[(None, 200)]	0	[]
input_7 (InputLayer)	[(None, 200)]	0	[]
lstm_4 (LSTM)	[(None, None, 200), (None, 200), (None, 200)]	320800	['embedding_1[2][0]', 'input_6[0][0]', 'input_7[0][0]']
lstm_5 (LSTM)	[(None, None, 200), (None, 200), (None, 200)]	320800	['lstm_4[3][0]']
input_8 (InputLayer)	[(None, 300, 200)]	0	[]
time_distributed (TimeDistribu ted)	(None, None, 3341)	671541	['lstm_5[1][0]']
=====			
Total params: 1,981,341			
Trainable params: 1,981,341			
Non-trainable params: 0			

Rysunek 3.5 Podsumowanie utworzonego dekodekera. Opracowanie własne

3.3.4 Trening

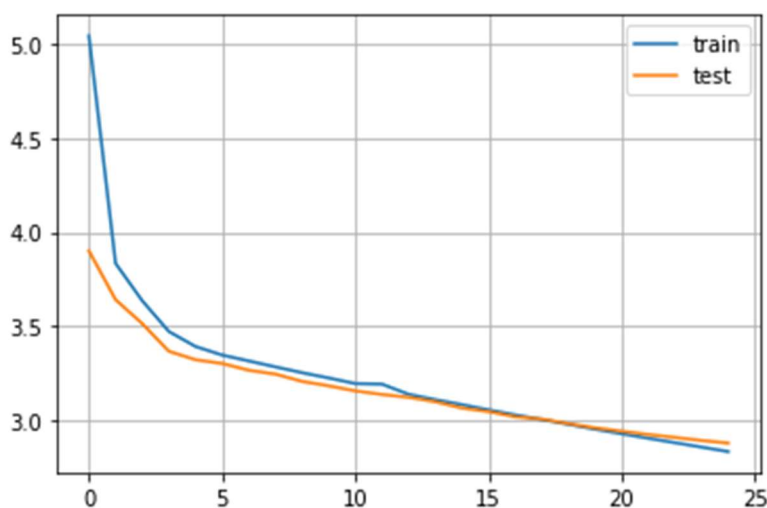
Do trenowania modelu wykorzystano funkcję straty z biblioteki Tensorflow „sparse_categorical_crossentropy”. Oparta jest ona na cross entropy loss wyrażonej wzorem:

$$\text{loss}_{c-e} = -\log P_{\theta}(\hat{y}|x) \quad (3.1)$$

Następnie rozpoczęto proces uczenia modelu, z możliwością wcześniejszego przerwania w sytuacji gdy miara jakości modelu na zbiorze walidacyjnym przestanie maleć. Od strony technicznej proces uczenia polegał na wykonywaniu predykcji streszczenia modelem, a następnie porównaniu wyniku ze znanym streszczeniem dla danego tekstu.

```
17/17 [=====] - 157s 9s/step - loss: 3.0278 - val_loss: 3.0176
Epoch 18/25
17/17 [=====] - 156s 9s/step - loss: 3.0030 - val_loss: 3.0058
Epoch 19/25
17/17 [=====] - 155s 9s/step - loss: 2.9775 - val_loss: 2.9810
Epoch 20/25
17/17 [=====] - 155s 9s/step - loss: 2.9516 - val_loss: 2.9588
Epoch 21/25
17/17 [=====] - 160s 9s/step - loss: 2.9272 - val_loss: 2.9412
```

Rysunek 3.6 Fragment przebiegu procesu uczenia. Opracowanie własne



Rysunek 3.7 Wykres jakości modelu na zbiorze treningowym i walidacyjnym. Opracowanie własne

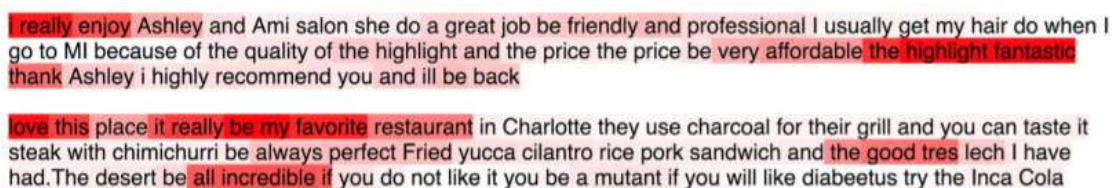
Pomimo licznych podejść do procesu uczenia wynik za każdym razem był podobny – streszczenia wszystkich tekstów brzmiały tak samo. W procesie uczenia wykorzystanym na potrzeby aplikacji wynikiem za każdym razem była fraza:

Predicted summary: gra przez deweloperską assembly

Przyczyn takiego stanu rzeczy można się doszukiwać w różnych miejscach, jednak najbardziej prawdopodobnym powodem jest zanikający gradient. Pomimo faktu, że warstwy LSTM zostały stworzone w celu rozwiązania problemu zanikającego gradientu, to w pracy przetwarzają długie sekwencje słów. Enkoder jak wyjaśniono w rozdziale 2.2 ma za zadanie zamienienie całej streszczanej sekwencji na wektor o zadanym rozmiarze, który następnie analizowany jest przez dekodery. Działanie takie można zinterpretować w taki sposób iż dekodery niejako „patrzy” na cały tekst skompresowany do stosunkowo małego rozmiaru. Sprawia to że znaczenie zawarte w tekście gubi się i algorytm stara się znaleźć takie zdanie które najlepiej odda wszystkie teksty w bazie danych jednocześnie.

Istnieją dwie główne metody rozwiązania tego problemu:

- Rozbudowa modelu do takiego stopnia, że wektor generowany przez enkoder będzie na tyle długi, że nie będzie „gubił” informacji
- Wprowadzenie mechanizmu uwagi, który będzie miał za zadanie wychwytywać najważniejsze fragmenty tekstu, a co za tym idzie niskim kosztem (brak konieczności rozbudowy modelu) ułatwi analizę dekoderni [19]



I really enjoy Ashley and Ami salon she do a great job be friendly and professional I usually get my hair do when I go to MI because of the quality of the highlight and the price the price be very affordable the highlight fantastic thank Ashley i highly recommend you and ill be back

love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had. The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola

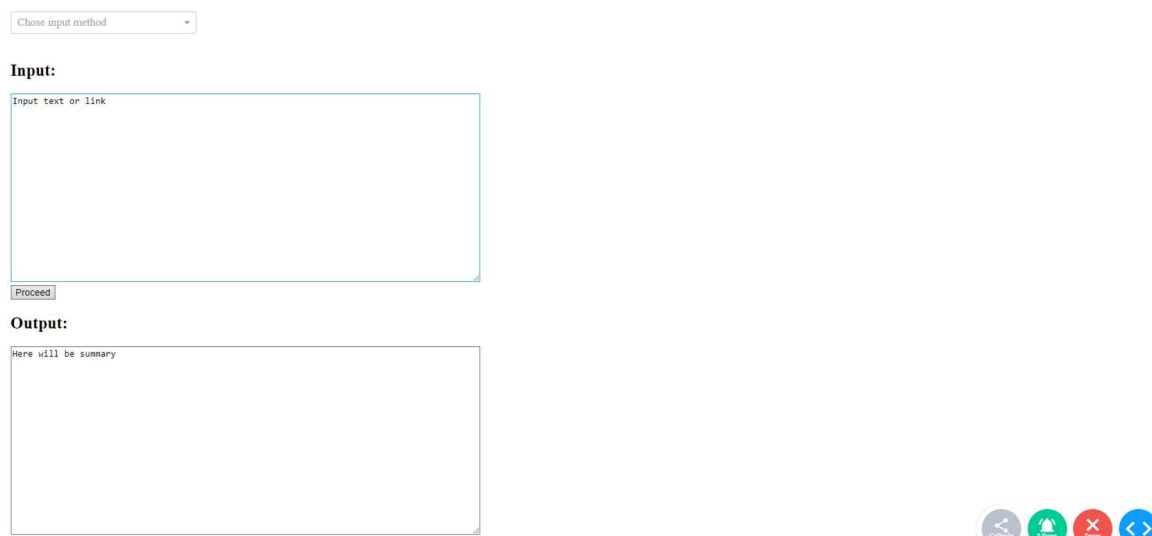
Rysunek 3.8 Przykładowa wizualizacja działania mechanizmu uwagi [20]

3.4 Tworzenie pozostałej części aplikacji

Ze względu na odmowę dostępu, z niewiadomych przyczyn, do API Twittera część aplikacji odpowiedzialną za umieszczanie wiadomości na portalu musiała zostać zrealizowana w niestandardowy sposób. Przy użyciu botów połączonych z komunikatorem Telegram została stworzona procedura pozwalająca publikować tweety przy pomocy specjalnie skonstruowanego linku. Po wygenerowaniu w aplikacji streszczenia jest ono publikowane na koncie @GatRaspberry

Dla pozostałej części aplikacji został stworzony interfejs, wraz z całą logiką odpowiedzialną za poprawne działanie

Automatic text summarization app for polish game descriptions



Rysunek 3.9 Wygląd utworzonej aplikacji. Opracowanie własne

Interfejs umożliwia wybranie czy tekst będzie wprowadzony ręcznie, czy podany zostanie link do strony na portalu gry-online.pl. Po wpisaniu odpowiedniej zawartości w pole Input i zatwierdzeniu przyciskiem Proceed w oknie Output pojawia się wygenerowane streszczenie, oraz treść jest publikowana na Twitterze, lub wyświetlona zostanie informacja o błędnie podanym linku. Po utworzeniu i przetestowaniu aplikacji lokalnie została ona wraz z wymaganymi do jej działania plikami umieszczona na platformie Heroku.

```
D:\AGH\Semestr_7\Praca_InzWeb_app_h>git push heroku master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 305 bytes | 305.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----> Building on the Heroku-20 stack
remote: ----> Using buildpack: heroku/python
remote: ----> Python app detected
remote: ----> Using Python version specified in runtime.txt
remote: ----> No change in requirements detected, installing from cache
remote: ----> Using cached install of python-3.8.12
remote: ----> Installing pip 21.3.1, setuptools 57.5.0 and wheel 0.37.0
remote: ----> Installing SQLite3
remote: ----> Installing requirements with pip
remote: ----> Discovering process types
remote: Procfile declares types -> web
remote:
remote: ----> Compressing...
remote: Done: 435M
remote: ----> Launching...
remote: ! Warning: Your slug size (435 MB) exceeds our soft limit (300 MB) which may affect boot time.
remote: Released v12
remote: https://my-app-inz.herokuapp.com/ deployed to Heroku
```

Rysunek 3.10 Fragment konsoli podczas wgrywania aplikacji na Heroku. Opracowanie własne

Automatic text summarization app for polish game descriptions

Link to gry-online ×

Input:

<https://www.gry-online.pl/gry/agatha-christie-hercule-poirot-the-first-cases/zf6018#pc>

Proceed

Output:

gra przez deweloperską assembly

Rysunek 3.11 Przykład działania aplikacji na Heroku



Rysunek 3.12 Tweet wygenerowany w wyniku działania przedstawionego na rysunku 3.11

3.5 Dokumentacja

Wykonanie dokumentacji zostało zrealizowane przy pomocy modułu pdoc [21], oraz utworzonych podczas pisania funkcji ich opisów w formie docstringów. Dzięki wykorzystaniu wspomnianego modułu uzyskano gotową stronę internetową zawierającą opisy wykorzystanych w projekcie funkcji.

Index

Super-module

python_functions

Functions

decode_sequence
prepare_text
text_cleaner

Module

python_functions.predicting

EXPAND SOURCE CODE

Functions

def decode_sequence(input_seq)

Predict summary for input

Args

input_seq : numpy.ndarray

tokenized text for summarization

Returns

str

Predicted summary for input

Rysunek 3.13 Jedna ze stron wykonanej dokumentacji

3.6 Testowanie

Celem przetestowania poprawności działania wgrywanych funkcji stworzono w oparciu o wbudowane funkcje GitHuba workflow mający na podstawie zadanych testów jednostkowych dokonywać sprawdzenia i wychwycenia ewentualnych błędów w implementacji oraz działaniu wgrywanego kodu.

```
1 ▶ Run pytest python_functions -vv --color=yes
7 ===== test session starts =====
8 platform linux -- Python 3.8.12, pytest-6.2.5, py-1.11.0, plugy-1.0.0 -- /opt/hostedtoolcache/Python/3.8.12/x64/bin/python
9 cachedir: .pytest_cache
10 rootdir: /home/runner/work/TweetBot/TweetBot
11 plugins: dash-2.0.0
12 collecting ... collected 3 items
13
14 python_functions/test_predicting.py::MyTestCase::test_prepare_text PASSED [ 33%]
15 python_functions/test_predicting.py::MyTestCase::test_text_cleaner PASSED [ 66%]
16 python_functions/test_webscrapper.py::MyTestCase::test_extract_desc_for_app PASSED [100%]
17
18 ===== warnings summary =====
19 ../../../../opt/hostedtoolcache/Python/3.8.12/x64/lib/python3.8/site-packages/flatbuffers/compat.py:19
20 /opt/hostedtoolcache/Python/3.8.12/x64/lib/python3.8/site-packages/flatbuffers/compat.py:19: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's
  documentation for alternative uses
21     import imp
22
23 -- Docs: https://docs.pytest.org/en/stable/warnings.html
24 ===== 3 passed, 1 warning in 2.96s =====
```

Rysunek 3.14 Raport potwierdzający spełnienie przez funkcje zadanych testów. Wygenerowany automatycznie po dokonaniu push'a - zgodnie z założeniami ciągłej integracji

4 Podsumowanie

4.1 Wnioski

Praca z językiem polskim w zagadnieniach NLP nie jest łatwym zadaniem i wymaga dobrej znajomości gramatyki i analizy języka, czego zabrakło w tej pracy. Wyniki sumaryzacji tekstu nie są zadawalające, jednak ograniczone zasoby sprzętowe i czas nie pozwoliły na przeanalizowanie innych rozmiarów modelu, czy podejść do problemu. Podczas realizacji części aplikacji odpowiedzialnej za umieszczanie wygenerowanych wiadomości na Twitterze napotkano problem w postaci odmowy dostępu do API platformy, przez co zaszła konieczność bardzo zawiłego publikowania wiadomości na Twitterze, jednak finalnie udało się osiągnąć zamierzony efekt. Pozostała część aplikacji również spełnia swoją funkcję i po odpowiednim ulepszeniu modelu jest w stanie realizować założoną funkcjonalność

4.2 Potencjalne kierunki rozwoju

Jako pole do poprawy działania stworzonego modelu, oraz aplikacji można wyróżnić następujące kierunki:

- Model dokonujący sumaryzacji – zastosowanie bardziej rozbudowanej sieci, lub skorzystanie z gotowych wstępnie wyuczonych modeli może poprawić wyniki. Również pozyskanie większej bazy danych, o bardziej zróżnicowanych tekstach może wpłynąć pozytywnie na proces uczenia modelu.
- Wstępne przetwarzanie tekstów – rozbudowanie tej funkcjonalności potencjalnie może ułatwić modelowi „rozumienie” przetwarzanego tekstu. Do możliwych do dodania metod przetwarzania można zaliczyć lematyzację – czyli sprowadzenie słów do ich formy podstawowej, oraz stemming – czyli usunięcie końcówki fleksyjnej z wyrazu, jednak bez odpowiednich badań trudno stwierdzić w jaki sposób wpłynie to na jakość sumaryzacji.
- Automatyzacja działania aplikacji – pozwoliłaby na umieszczanie wygenerowanych wiadomości na Twitterze w momencie pojawienia się na zadanej stronie nowego artykułu.

Bibliografia

- [1] Glenc, P. (2021). Narzędzia do automatycznego streszczania tekstów w języku polskim. Stan badań naukowych i prac wdrożeniowych. e-mentor, 2(89), 67-77.
- [2] Chopra, A., Prashar, A., & Sain, C. (2013). Natural language processing. International journal of technology enhancements and emerging engineering research, 1(4), 131-134.
- [3] Grafika. [Online]. [dostęp: 8 styczeń 2022] https://medium.com/@rimacyn_23654/auto-highlighter-extractive-text-summarization-with-sequence-to-sequence-model-cbbf333772bf
- [4] Tian, S, Yaser, K, Naren, R, Chandan, K. (2021) Neural Abstractive Text Summarization with Sequence-to-Sequence Models. ACM/IMS Trans. Data Sci. 2, 1, Article 1
- [5] Dokumentacja pakietu Dash. [Online]. [dostęp: 8 styczeń 2022]. <https://dash.plotly.com/>
- [6] Dokumentacja platformy Heroku. [Online]. [dostęp: 8 styczeń 2022]. <https://devcenter.heroku.com/categories/reference>
- [7] Dokumentacja pakietu TensorFlow. [Online]. [dostęp: 8 styczeń 2022]. https://www.tensorflow.org/api_docs/python/tf/all_symbols
- [8] Dokumentacja pakietu Keras. [Online]. [dostęp: 8 styczeń 2022]. <https://keras.io/api/>
- [9] Dokumentacja pakietu NumPy. [Online]. [dostęp: 8 styczeń 2022]. <https://numpy.org/doc/stable/reference/index.html#>
- [10] Dokumentacja Google Colaboratory. [Online]. [dostęp: 8 styczeń 2022]. https://colab.research.google.com/notebooks/basic_features_overview.ipynb
- [11] Dokumentacja pakietu BeautifulSoup. [Online]. [dostęp: 8 styczeń 2022]. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- [12] Dokumentacja pakietu Selenium. [Online]. [dostęp: 8 styczeń 2022]. <https://selenium-python.readthedocs.io/index.html>

- [13] Dokumentacja pakietu Requests. [Online]. [dostęp: 8 styczeń 2022].
<https://docs.python-requests.org/en/master/user/quickstart/>
- [14] Repozytorium projektu na GitHubie [Online]. [dostęp: 11 styczeń 2022].
<https://github.com/KAIR-ISZ-NLP/TweetBot>
- [15] Portal z którego pozyskano zbiór danych [Online]. [dostęp: 8 styczeń 2022].
<https://www.gry-online.pl>
- [16] Model o który oparto ten stworzony w projekcie [Online]. [dostęp: 8 styczeń 2022].
<https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/>
- [17] Lista stop words wykorzystana w projekcie [Online]. [dostęp: 8 styczeń 2022].
<https://github.com/bieli/stopwords/blob/master/polish.stopwords.txt>
- [18] Pojęcie stop words [Online]. [dostęp: 8 styczeń 2022].
<https://pl.wikipedia.org/wiki/Wikipedia:Stopwords>
- [19] Niu, Z., Zhong, G., & Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, 452, 48–62.
- [20] Lin, Z., Feng, M., Santos, C.N., Yu, M., Xiang, B., Zhou, B., & Bengio, Y. (2017). A Structured Self-attentive Sentence Embedding. *ArXiv*, abs/1703.03130.
- [21] Dokumentacja generatora pdoc3. [Online]. [dostęp: 8 styczeń 2022].
<https://pdoc3.github.io/pdoc/>