Primjena tehnika za analizu i prepoznavanje jezika na tekstualnim vijestima

Majetić, Domagoj

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Domagoj Majetić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Domagoj Majetić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru doc. dr. sc. Tomislavu Stipančiću na pruženoj pomoći te uloženom trudu i vremenu tijekom izrade ovog rada.

Posebno se zahvaljujem svojoj obitelji na podršci i razumijevanju te svim prijateljima koji su mi uljepšali dosadašnji dio studija.

Domagoj Majetić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Domagoj Majetić** JMBAG: **0035215314**

Naslov rada na hrvatskom jeziku: **Primjena tehnika za analizu i prepoznavanje jezika na tekstualnim vijestima**

Naslov rada na engleskom jeziku: **Application of language analysis and recognition techniques to a textual news feed**

Opis zadatka:

Računalna analiza i prepoznavanje govora (eng. Natural Language Processing) je sposobnost računalnog agenta ili softvera da prepoznaje riječi i izraze u jeziku te ih analizira ili pretvara u ljudima čitljivi tekst. Programske aplikacije temeljene na NLP-u su brojne, te uključuju analizu osjećaja, izradu komunikacijskih agenata (eng. Chatbots), glasovno pretraživanje teksta, prepoznavanje govora, filtriranje elektroničke pošte, i dr.
RSS tekst (eng. RDF Site Summary ili Really Simple Syndication – RSS feed) predstavlja sažetak neke veće količine teksta koji se kao vijest može prikazati unutar neke aplikacije ili druge web stranice. Pomoću RSS teksta je moguće na jednom mjestu stvoriti novu vijest koja u sebi sadrži informacije od nekoliko različitih web izvora.
U radu je potrebno razviti programsku aplikaciju koja u sebi ima integrirano barem tri proizvoljne NLP tehnike. Aplikaciju je potrebno generirati tako da se automatski puni novim tekstom koristeći vijesti iz različitih izvora objavljenih na webu (npr. NY Times, CNN i sl.).
Koristeći odabrane NLP tehnike u sljedećem koraku je potrebno analizirati tekst nastao kao rezultat primjene RSS teksta čime je moguće saznati dodatne informacije o prikupljenim vijestima na automatizirani način. Rad treba sadržavati opis odabranih NLP tehnika te kritički osvrt na učinke njihove uporabe.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
9. 5. 2022.	**2. rok (izvanredni):** 6. 7. 2022. **3. rok:** 22. 9. 2022.	**2. rok (izvanredni):** 8. 7. 2022. **3. rok:** 26. 9. – 30. 9. 2022.
Zadatak zadao:		Predsjednik Povjerenstva:
Doc. dr. sc. Tomislav Stipančić		Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
S	/	Skup mogućih opažanja
P	/	Skup distribucija vjerojatnosti na S
Θ	/	Skup parametara modela
x	/	Ulazni signal
w	/	Težinski faktor
y	/	Izlazni signal
f	/	Aktivacijska funkcija

SAŽETAK

Tema ovog rada je razvoj programske aplikacije koja pomoću integriranih NLP tehnika analizira sadržaj članaka novinskih portala. Razvijena programska aplikacija bazira se na programskom jeziku Python. Program funkcionira pomoću implementiranih biblioteka koje omogućavaju uporabu tehnika obrade prirodnog jezika (NLP) i RSS teksta koji se preuzima sa web stranica novinskih portala. U prvom dijelu rada razrađena je teorijska osnova i razvoj računalne analize i prepoznavanja govora (eng. NLP - *Natural Language Processing*). Poznavanjem teorijske osnove omogućava se razumijevanje razvoja programske aplikacije. U drugom dijelu prikazana je izvedba programa u programskom jeziku Python te detaljan opis i kritički osvrt na učinke uporabe odabranih NLP tehnika.

Ključne riječi: NLP, RSS tekst, Python, analiza sadržaja članaka novinskih portala

SUMMARY

The topic of this paper is the development of the software that analyzes articles of news portals using integrated NLP techniques. The developed software is based on the Python programming language. The software functions by using imported libraries enabling the use of natural language processing techniques (NLP) and RSS feed that is downloaded from the news portal websites. The theoretical basis and development of Natural Language Processing is explained in the first part of this paper. By knowing the theoretical basis, it is possible to understand how the software works. The second part of the paper shows the development of news articles analysis software with a detailed description and a critical review of the effects of using selected NLP techniques.

Key words: NLP, RSS feed, Python, news articles analysis software

1. UVOD

Umjetna inteligencija bavi se razvojem sposobnosti računala da obavljaju zadatke za koje je potreban neki oblik inteligencije. U skladu sa napretkom umjetne inteligencije dolazi do pojave i razvoja novih metoda i komponenti koje se vežu na taj dio računalne znanosti. Jedna od tih komponenti je i računalna analiza i prepoznavanje govora (eng. NLP - *Natural Language Processing*) koja će biti detaljnije opisana u nastavku ovog rada. NLP omogućava računalu da razumije tekst ili govor stvoren od strane čovjeka. Kao što ljudi imaju mozak za procesuiranje takvih informacija, računalima se pomoću programa daje mogućnost da na sličan način obrađuju ulazne podatke.

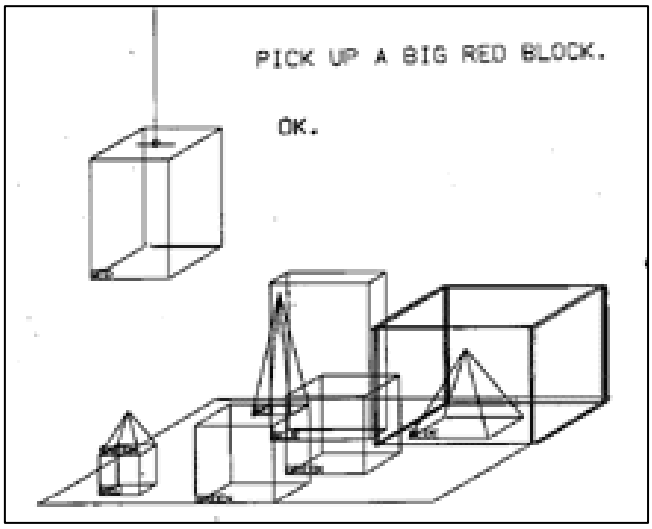
Jedna od primjene NLP-a je analiza i obrada teksta. U programskoj aplikaciji razvijenoj u sklopu ovoga rada taj tekst se učitava u obliku RSS teksta. RSS tekst (eng. RSS feed – *Really Simple Syndication*) predstavlja sažetak neke veće količine teksta koji se u konkretnom slučaju preuzima sa određenih novinskih portala. Programska aplikacija preuzima RSS tekst, zatim ga odabranim NLP tehnikama obrađuje i na kraju stvara dodatne informacije o prikupljenim vijestima na automatizirani način. Te informacije koje su u konačnici pružene korisniku mogu se dalje koristiti za razne analize ili stvaranje statistike.

2. RAČUNALNA ANALIZA I PREPOZNAVANJE GOVORA

2.1. Razvoj računalne analize i prepoznavanja govora

2.1.1. Simbolički NLP

Ova vrsta NLP-a pojavila se početkom 1950ih te je bila aktualna do ranih 1990ih. Ideja iza simboličkog NLP-a je bila naučiti računalo kako razumjeti jezike na isti način kako je čovjek naučio čitati i pisati. Neki od uspješnijih sustava računalne analize i prepoznavanja govora razvijeni su 1960ih, SHRDLU je bio jedan od najranijih računalnih programa sposobnih za interakciju između čovjeka i računala, prikaz njegovog korištenja prikazan je na slici [Slika 1.]. Nakon toga dolazi do pojave prvih „chatbotova“, softvera koji omogućavaju automatiziranu komunikaciju čovjeka i računala. Računalo se prilagođava porukama koje dobiva od korisnika te smisleno odgovara na njih [1].



Slika 1. Prikaz korištenja programa SHRDLU [3]

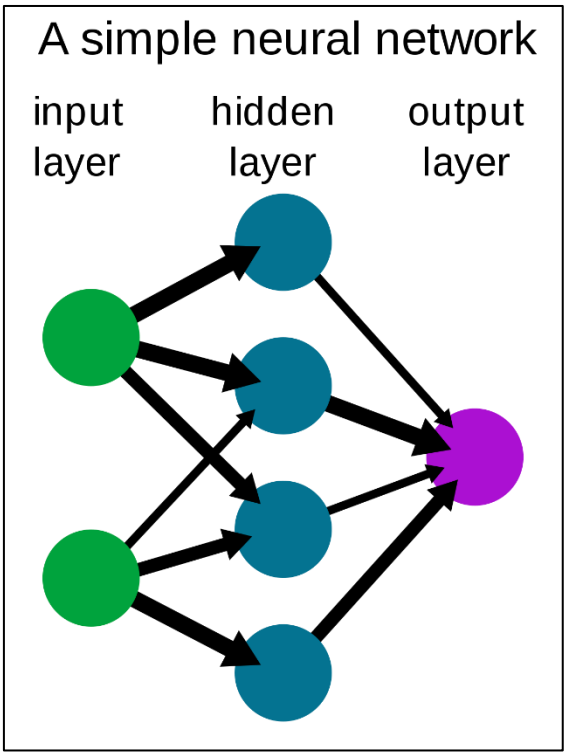
2.1.2. Statistički NLP

Do 1980ih većina NLP sustava bili su bazirani na kompleksnim setovima pravila koje je unosio čovjek. To se promijenilo početkom 1990ih pojavom statističkog NLP-a i strojnog učenja [1]. Strojno učenje je vrsta učenja u sklopu umjetne inteligencije koje omogućava računalima da budu što precizniji u predviđanju i donošenju odluka bez da su nužno programirani za takve radnje [2]. Statističkim NLP-om dolazi do pojave automatizirane obrade prirodnog jezika što

je omogućilo računalu da čita i razumije značenje ljudskih jezika. Više o primjeni statističkih metoda detaljnije je opisano u nastavku ovog rada.

2.1.3. Neuronski NLP

U 21. stoljeću metode dubokog učenja postaju sve raširenije u području obrade prirodnog jezika. Duboko učenje je oblik strojnog učenja u kojemu model koji se obučava ima više slojeva između ulaza i izlaza, ti slojevi su implementirani u obliku neuronskih mreža koje su detaljnije opisane u sljedećim poglavljima, shematski prikaz neuronske mreže prikazan je na slici [Slika 2.] [4]. Primjena takvog oblika učenja i neuronskih mreža bila je od velikog značaja u području medicine i zdravstvene skrbi, gdje se uz pomoć NLP-a pomoglo analizirati bilješke i zapise o bolestima u elektroničkim zdravstvenim kartonima pacijenata [1].



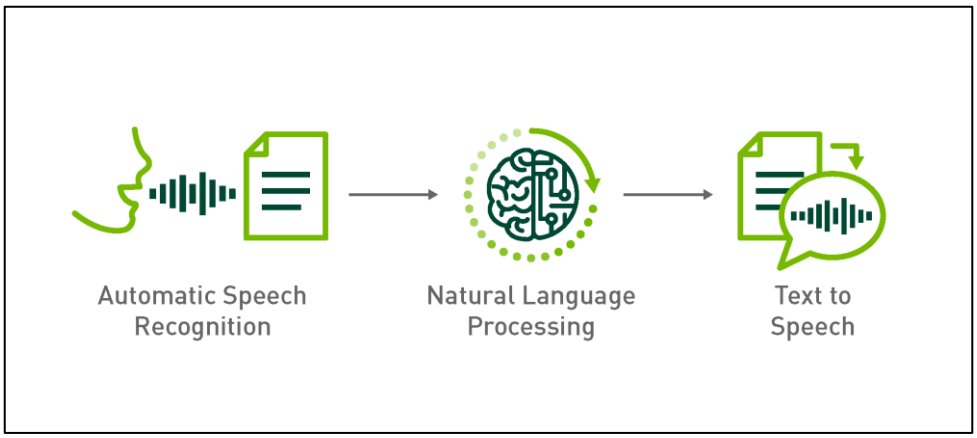
Slika 2. Shematski prikaz neuronske mreže [5]

2.2. Metode obrade prirodnog jezika

2.2.1. Statističke metode i modeli

Statistički model je matematički model koji objedinjuje skup statističkih pretpostavki s obzirom na generirajući uzorak podataka. Takav model predstavlja proces generiranja podataka u idealiziranom obliku. Obično se identificira kao matematički odnos između jedne ili više slučajnih varijabli i drugih neslučajnih varijabli. Kao takav može se reći da je statistički model „formalni prikaz teorije“ odnosno statistički modeli temelj su statističkog zaključivanja. Model se može matematički prikazati kao par (S, P), gdje S predstavlja skup mogućih opažanja, a P označava skup distribucija vjerojatnosti na S. $P = \{P_\theta : \theta \in \Theta\}$, gdje skup Θ definira parametre modela [6].

Takvi modeli imaju prednost jer mogu prikazati više mogućih rješenja umjesto samo jednog, dajući pouzdanije rezultate kada je takav model uključen kao komponenta većeg sustava. Statistički modeli odnosno statistička metoda NLP-a primjenjuje se najčešće u sustavima za prepoznavanje glasa ili govora. To su sustavi koji omogućuju računalu da prepozna i prevede jezik koji govori čovjek kao tekstualni zapis, shematski prikaz uporabe sustava za prepoznavanje govora prikazan je na slici [Slika 3.]. Neki sustavi za prepoznavanje glasa zahtijevaju trening gdje korisnik čita tekst ili određene riječi iz vokabulara i na taj način ih pohranjuje u sustav [7]. Od pojave neuronskog NLP-a, statističke metode u NLP sustavima uvelike su zamijenjene neuronskim mrežama, no unatoč tome i dalje su relevantne za kontekste u kojima je potrebna statistička interpretabilnost i transparentnost.



Slika 3. Shematski prikaz uporabe sustava za prepoznavanje govora [27]

Još neke od primjena statističkog NLP-a:

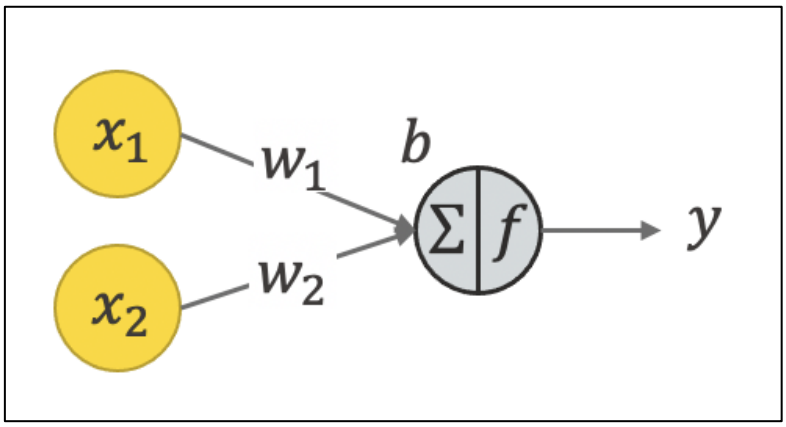
- Predlaganje riječi prilikom pisanja teksta,
- Prepoznavanje rukopisa unatoč nečitko napisanom tekstu,
- Uočavanje i ispravljanje gramatičkih grešaka,

- Sažimanje teksta i
- Kategorizacija teksta [8].

2.2.2. Neuronske mreže

Umjetna neuronska mreža predstavlja skup umjetnih neurona koji su međusobno povezani i interaktivni kroz operacije obrade signala. Svrha umjetne neuronske mreže je da oponaša ljudski mozak. Mreža ima jedan ili više ulaza te jedan izlaz između kojih se nalazi jedan ili više skrivenih slojeva, takve mreže još se nazivaju višeslojne mreže [9].

Signal je opisan numeričkom vrijednošću. Na ulazu neurona, signale (x_1, x_2) potrebno je pomnožiti težinskim faktorom (w_1, w_2) koji daju informaciju o jakosti sinapse tj. spoju dva neurona. Pomnoženi signali se sumiraju, a na tu sumu se primjenjuje aktivacijska funkcija f koja daje izlazni signal y. Prijenos signala prikazan je na slici [Slika 4.]

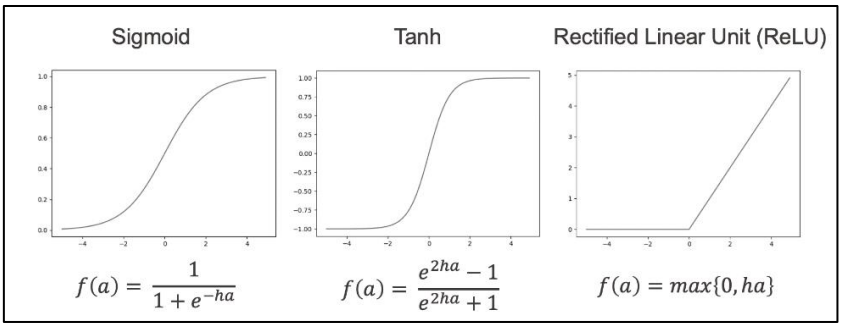


Slika 4. Građa umjetnog neurona [10]

Postoji mogućnost odabira vrste aktivacijske funkcije koja će se koristiti prilikom modeliranja neuronske mreže. Neke od češće korištenih funkcija su:

- Sigmoidna funkcija,
- Hiperbolična tangentna funkcija i
- ReLU prijenosna funkcija.

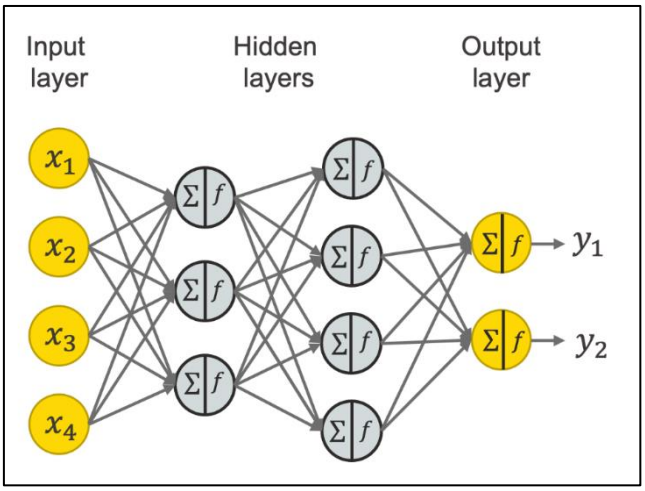
Izgled navedenih funkcija prikazan je na slici [Slika 5.]



Slika 5. Prikaz aktivacijskih funkcija [10]

Kombinacijom većeg broja neurona stvara se neuronska mreža. Način na koji se neuroni povezuju u neuronsku mrežu prikazan je na slici [Slika 6.]. Neuronska mreža sastoji se od ulaznog sloja, izlaznog sloja te jednog ili više skrivenih slojeva. Svi neuroni osim ulaznih imaju vezu s neuronima iz prethodnog sloja [10].

Glavna primjena umjetnih neuronskih mreža je kod traženja zavisnosti između podataka koji nisu u isključivo linearnoj vezi, ali se mogu ujediniti u jedan složeni ulazni skup. Ujedinjavanjem takvih podataka bave se stručnjaci u području u kojem se neuronske mreže primjenjuju. Temeljno obilježje svih mreža, bez obzira na oblik i broj veza unutar njih jest da se odlikuju svojstvom „učenja", tj. uvježbavanja kroz niz ponavljajućih postupaka analize. Od cijeloga skupa podataka veći dio upotrijebljen je za učenje, a manji za ponovno predviđanje poznatih vrijednosti. Na taj način moguće je izračunati pogrešku predviđanja, koja bi s većim brojem pokušaja trebala biti manja. Takav tip „učenja" podsjeća na ljudsko učenje iz iskustva pa odatle i naziv „neuronske mreže" [9].



Slika 6. Model neuronske mreže [10]

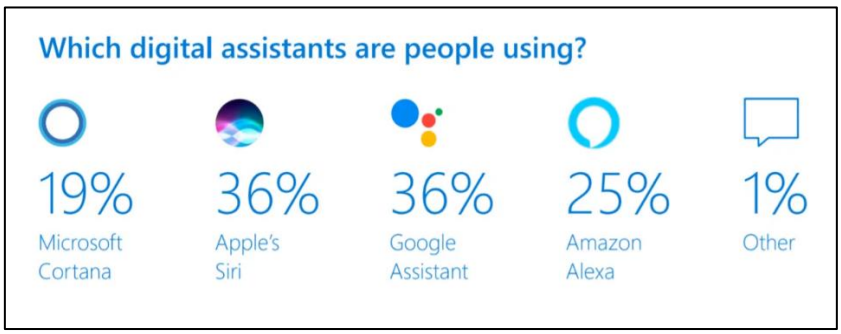
2.3. Primjene NLP-a u programskim aplikacijama

Neke od značajki programskih aplikacija koje funkcioniraju pomoću NLP tehnika su:

- Analiza sentimenta,
- Sažimanje teksta,
- Pretraživanje dokumenata,
- Izdvajanje teksta,
- Prepoznavanje govora,
- Automatsko ispravljanje pogrešno napisanih riječi,
- Automatsko popunjavanje rečenica i
- Prevođenje tekstova.

Jedna od najzastupljenijih primjena NLP-a je u području virtualnih asistenata, postoji mnogo vrsta različitih virtualnih asistenata koji su implementirani u raznim operativnim sustavima. Neki od poznatijih virtualnih asistenata su Microsoftova Cortana, Appleov Siri, Googleov asistent te Amazonova Alexa.

Virtualni asistenti imaju mogućnost prepoznavanja čovjekovog glasa. Čovjek daje naredbe virtualnom asistentu poput postavljanja podsjetnika, otvaranje aplikacija, slanje mailova, praćenja letova i pošiljki, provjere vremenske prognoze i slično, nakon što virtualni asistent primi te naredbe on ih u realnom vremenu izvršava. Na slici [Slika 7.] prikazana je usporedba korištenosti raznih virtualnih asistenata u svijetu.



Slika 7. Prikaz usporedbe korištenosti virtualnih asistenata [11]

3. IZVEDBA ZADATKA U PYTHONU

U sljedećim poglavljima razrađen je praktični dio rada odnosno izvedba zadanog zadatka u programskom jeziku Python. U prvom dijelu prikazan je opis rada programske aplikacije. U drugom dijelu obrađena je teorijska osnova programskog jezika Python i neke od biblioteka koje su korištene tokom izrade ovog zadataka. Na kraju je detaljno razrađena funkcionalnost programa od načina preuzimanja tesktualnih informacija s novinskih članaka do analize svih NLP tehnika koje su korištene.

3.1. Programska aplikacija za analizu članaka novinskih portala

Programska aplikacija za analizu članaka novinskih portala (eng. *News Articles Analysis application*) je aplikacija kojoj je cilj da iz web stranica novinskih portala prikuplja sve tekstualne informacije koje se nalaze u člancima, od naslovnica, opisa članaka pa sve do samog sadržaja tih članaka. Nakon što aplikacija preuzme sav tekstualan sadržaj iz članaka, uz pomoću NLP tehnika, analizira i obrađuje te podatke. Glavni zadatak aplikacije je da te podatke na sistematiziran način rasporedi i prikaže korisniku ovisno o NLP tehnici koju izabere. Te podatke korisnik može koristiti za daljnje analize i statistike ili za lakši i brži pristup člancima bez da nužno otvara portal i prolazi kroz čitav sadržaj web stranice.

3.2. Programski jezik Python

Python je interpretacijski programski jezik opće namjene i visoke razine. Dopušta programerima korištenje više stilova programiranja – objektno, strukturno i aspektno orijentirano programiranje. Zbog te raznolikosti stilova programiranja te izražene jednostavnosti i intuitivnosti kod korištenja, postaje sve popularniji među korisnicima diljem svijeta. Python se svrstava u top 10 najpopularnijih programskih jezika u Indeksu programske zajednice TIOBE, gdje je od 2021. postao i najpopularniji jezik [12].

U ovom radu korišten je Python 3.9.7., inačica Pythona koja sadrži većinu potrebnih biblioteka koje su korištene prilikom razvoja programske aplikacije i implementacije NLP tehnika.

Biblioteke koje nisu dio ove verzije Pythona instalirane su pomoću „pip install“ komande u terminalu. Više o bibliotekama razrađeno je u nastavku rada.

3.2.1. Python biblioteke i paketi

Prilikom izrade programske aplikacije, na početku koda programa, bilo je potrebno implementirati sve potrebne biblioteke i pakete korištene za funkcionalnost odabranih NLP tehnika. Taj dio koda prikazan je na slici [Slika 8.].

```
#Import all the required livraries..

import pandas as pd
import feedparser
# Import packages
import matplotlib.pyplot as plt
#get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import gensim
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import word_tokenize
import pyLDAvis.gensim_models
from collections import Counter
import feedparser
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import streamlit as st
import base64
import seaborn as sns
from PIL import Image
import cufflinks
from sklearn.feature_extraction.text import CountVectorizer
from plotly.offline import iplot
from textblob import TextBlob
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import plotly.express as px
import spacy
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.manifold import TSNE
#from gensim.summarization import summarize
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.lex_rank import LexRankSummarizer
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.lsa import LsaSummarizer as Summarizer
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.luhn import LuhnSummarizer
```

Slika 8. Implementiranje biblioteka i paketa u kodu

U sljedećim naslovima opisane su neke od biblioteka koje su implementirane te njihova svrha.

3.2.1.1 Pandas

Pandas je biblioteka stvorena za Python programski jezik koja se koristi za obradu i analizu podataka. Pruža strukture podataka i operacije za manipulaciju numeričkim tablicama. Naziv „Pandas“ je izveden iz izraza „Panel data“.

Neke od značajki Pandas biblioteke su:

- „DataFrame“ za manipulaciju podacima s integriranim indeksiranjem,
- Usklađivanje podataka i integrirano rukovanje podacima koji nedostaju,
- Preoblikovanje skupova podataka,
- Spajanje i združivanje setova podataka i
- Filtriranje podataka.

Pandas se uglavnom koristi za analizu, spajanje ili preoblikovanje podataka, podaci se mogu uvesti iz raznih izvora i tipova datoteka [13].

3.2.1.2 Feedparser

Feedparser je jednostavan, ali moćan Python paket koji se koristi za izvlačenje informacija o određenim web stranicama pomoću RSS teksta tih stranica. Korištenjem Feedparser paketa možemo dobiti strukturirane informacije u obliku Python listi ili riječnika. Te informacije se zatim raščlanjuju ili izvlače. Na primjer, uz pomoću Feedparser paketa može se pristupiti najnovijim objavama s neke web stranice. Nadalje, može se pristupiti različitim atributima poput linkova, slika, naslova ili opisa u obliku parova ključ-vrijednost (eng. *key-value pairs*) [14].

3.2.1.3 NumPy (Matplotlib, seaborn)

NumPy je Python biblioteka koja se koristi za rad s višedimenzijskim poljima (eng. *array*). Također koristi se za rad u području linearne algebre, Fourierovih transformacija te za operacije s matricama. Izraz NumPy definiran je kao kratica za Numerički Python (eng. *Numerical Python*) [15].

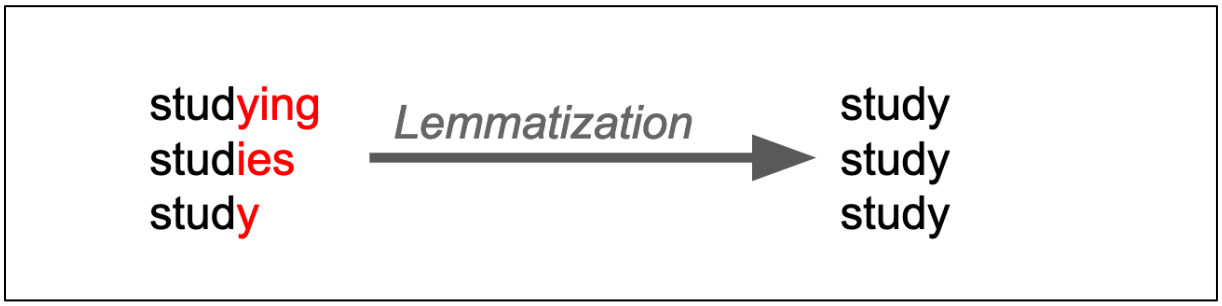
Matplotlib je biblioteka implementirana u obliku matematičkog proširenja NumPy biblioteke. Koristi se za crtanje grafova te njihovo ugrađivanje u programske aplikacije. Uz Matplotlib moguće je implementirati i seaborn. To je sučelje visoke razine za crtanje statističke grafike. Cilj mu je učiniti vizualizaciju glavnim faktorom istraživanja i razumijevanja složenih skupova podataka [16].

3.2.1.4 NLTK

NLTK (eng. *Natural Language Toolkit*) je niz biblioteka i programa za simboličko i statističko obrađivanje pomoću programskog jezika Python. NLTK je pretežito namijenjen učenju o računalnoj obradi prirodnog jezika. Neke od biblioteka koje sadrži su biblioteke za pretvaranje govora u tekst, prepoznavanje imena, antonima i sinonima, analizu emocija, odvajanje riječi i slično [17].

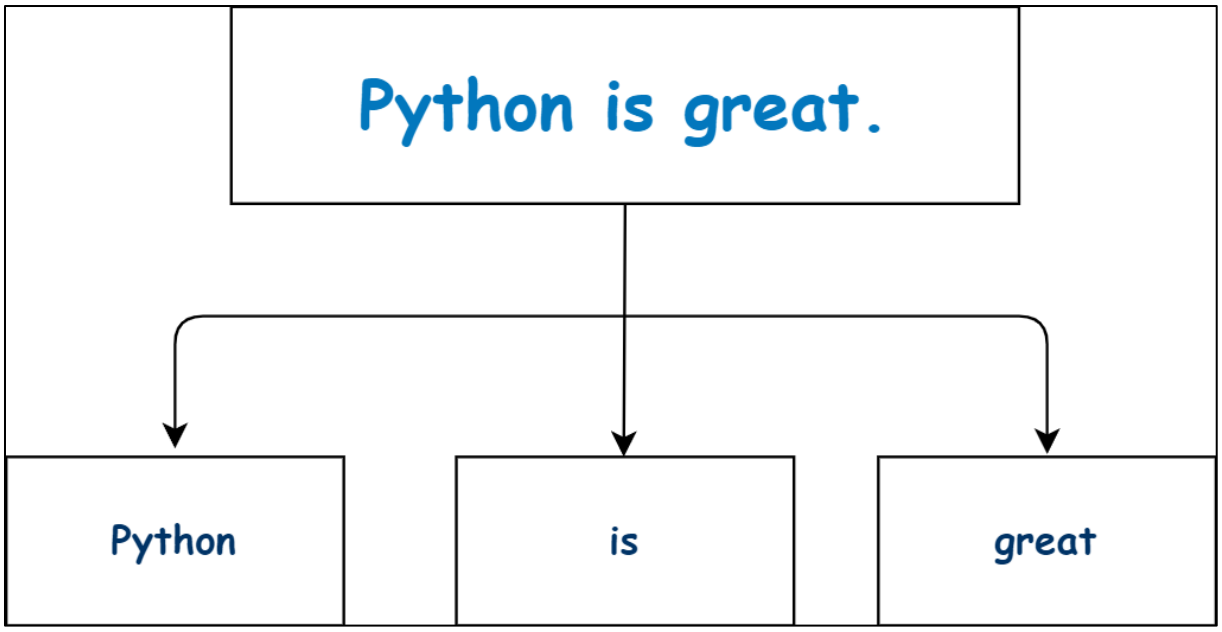
U okviru NLTK-a mogu se implementirati razni postupci obrade teksta kao što su WordNetLemmatizer i word_tokenize.

Lematizacija je proces grupiranja izoliranih oblika riječi na način da se svaka analizira zasebno kao posebna riječ. Pojedinačnim riječima se lematizacijom daje određeni kontekst. WordNetLemmatizer je leksička baza engleskog jezika kojoj je svrha uspostavljanje semantičkih veza između riječi. Na slici [Slika 9.] prikazan je primjer lematizacije 3 riječi [18].



Slika 9. Primjer lematizacije [28]

Tokenizacija je postupak kojim se rečenice dijele na riječi od kojih su sačinjene. Word_tokenize je funkcija u Pythonu koja izdvaja riječi iz rečenice u sklopu NLTK biblioteke. Na slici [Slika 10.] je prikazan primjer tokenizacije jedne rečenice [19].



Slika 10. Primjer tokenizacije

3.2.1.5 Spacy

Spacy je biblioteka koja se koristi u programskom jeziku Python za prirodnu obradu jezika, sačinjena je od mnogo ugrađenih mogućnosti. Postaje sve popularnija za obradu i analizu podataka u sklopu NLP-a. Primarna svrha biblioteke je obrada i izvlačenje nestrukturiranih tekstualnih podataka [20].

3.3. Funkcionalnost programa i odabrane NLP tehnike

3.3.1 Prikupljanje potrebnih tekstualnih informacija

RSS je skup web formata koji korisnicima ili aplikacijama omogućuje pristup ažuriranjima web stranica u standardiziranom formatu čitljivom za računala. Pomoću RSS-a moguće je pratiti mnogo različitih web stranica na jednom mjestu. Za funkcionalnost ove programske aplikacije bilo je potrebno pronaći URL link koji nas povezuje sa RSS tekstom željenih web stranica odabranih novinskih portala. Na temelju dobivenih RSS poveznica prikupljaju se potrebne tekstualne informacije s novinskih portala koje će se kasnije obrađivati različitim NLP tehnikama. Informacije koje se prikupljaju su:

- Naslovi članaka (eng. *Title*),

- Poveznice od članaka (eng. *Link*) i
- Opis članaka (eng. *Description*).

Prikupljene informacije se zatim unose u pandas data frame za njihovo lakše procesuiranje u daljnjem tijeku razvoja programske aplikacije. Na slici [Slika 11.] prikazan je dio koda koji je zadužen za prijenos dobivenih informacija u pandas data frame uz pomoću feedparser biblioteke.

```
# In[ ]:
class RSSFeed():
    feedurl = ""

    global ndf
    def __init__(self, paramrssurl):
        print(paramrssurl)
        self.feedurl = paramrssurl
        self.parse()


    def parse(self):
        thefeed = feedparser.parse(self.feedurl)
        global ndf
        ndf = pd.DataFrame(columns=['title', 'link', 'decription'])
        for thefeedentry in thefeed.entries:
            title = thefeedentry.get("title", "")
            link = thefeedentry.get("link", "")
            decr = thefeedentry.get("description", "")
            ndf = ndf.append({'title': title, 'link': link, 'decription': decr},
                             ignore_index=True)
        return ndf
```

Slika 11. Prijenos tekstualnih informacija u pandas data frame

Uz prikupljene informacije o naslovu, poveznici i opisu raznih članaka sa web stranica novinskih portala bilo je potrebno preuzeti i informacije o samom sadržaju tih članaka. Sadržaj članaka (eng. *Content*) dobiven je pomoću BeautifulSoup biblioteke. BeautifulSoup je Python biblioteka koja se koristi za izvlačenje podataka iz HTML i XML datoteka. Biblioteka stvara konkretno sintaksno stablo (eng. *parse tree*) iz izvornog koda stranice koje se može koristiti za izdvajanje podataka na hijerarhijski i čitljiviji način [21]. Na slici [Slika 12.] prikazan je dio koda u kojemu se uz pomoću BeautifulSoup biblioteke prikuplja sadržaj članaka.

```
#Beautiful Soup Code
@st.cache
def full_text(my_url):
    import requests
    from bs4 import BeautifulSoup
    import pandas as pd
    url = my_url
    article = requests.get(url)
    articles = BeautifulSoup(article.content, 'html.parser')
    articles_body = articles.findAll('body')
    p_blocks = articles_body[0].findAll('p')
    p_blocks_df = pd.DataFrame(columns=['element_name', 'parent_hierarchy', 'element_text', 'element_text_Count'])
    for i in range(0, len(p_blocks)):
        parents_list = []
        for parent in p_blocks[i].parents:
            Parent_id = ''
            try:
                Parent_id = parent['id']
            except:
                pass
            parents_list.append(parent.name + 'id: ' + Parent_id)
        parent_element_list = ['' if (x == 'None' or x is None) else x for x in parents_list]
        parent_element_list.reverse()
        parent_hierarchy = ' -> '.join(parent_element_list)
        p_blocks_df = p_blocks_df.append({"element_name": p_blocks[i].name
                                          , "parent_hierarchy": parent_hierarchy
                                          , "element_text": p_blocks[i].text
                                          , "element_text_Count": len(str(p_blocks[i].text))}
                                        , ignore_index=True
                                        , sort=False)
    if len(p_blocks_df) > 0:
        p_blocks_df_groupby_parent_hierarchy = p_blocks_df.groupby(by=['parent_hierarchy'])
        p_blocks_df_groupby_parent_hierarchy_sum = p_blocks_df_groupby_parent_hierarchy[['element_text_Count']].sum()
        p_blocks_df_groupby_parent_hierarchy_sum.reset_index(inplace=True)
    maxid = p_blocks_df_groupby_parent_hierarchy_sum.loc[
        p_blocks_df_groupby_parent_hierarchy_sum['element_text_Count'].idxmax()
        , 'parent_hierarchy']
    merge_text = '\n'.join(p_blocks_df.loc[p_blocks_df['parent_hierarchy'] == maxid, 'element_text'].to_list())
    return merge_text
```

Slika 12. Prikupljanje sadržaja članaka pomoću BeautifulSoup biblioteke

Nakon što su prikupljene sve potrebne informacije sa web stranica novinskih portala njih se može obrađivati i analizirati zadanim NLP tehnikama koje su obrađene u sljedećim poglavljima. Prilikom pokretanja programa, na padajućem izborniku, moguće je odabrati „Snapshot“ opciju koja prikazuje tablicu koja sadrži sve gore navedene prikupljene informacije. Prikaz informacija u programskoj aplikaciji prikazan je na slici [Slika 13.].

Slika 13. Prikaz prikupljenih informacija u programskoj aplikaciji

3.3.2 N-grams

„N-grams“ predstavlja slijed N broja riječi koje su kontekstualno povezane i pojavljuju se više puta u nekom tekstualnom zapisu. U ovoj programskoj aplikaciji NLP tehnika „N-grams“ funkcionira na način da se iz svih prikupljenih informacija preuzetih sa web stranica novinskih portala izabiru riječi koje se najviše ponavljaju u oblik jedne riječi (eng. *unigrams*), u obliku 2 povezane riječi (eng. *bigrams*) te u obliku 3 povezane riječi (eng. *trigrams*) [22].

Neke od primjena „N-grams“ u NLP-u su sljedeće:

- Automatsko popunjavanje rečenica,
- Automatsko prepoznavanje nepravilno napisanih riječi,
- Provjera gramatičke točnosti i
- Virtualni asistenti – botovi.

Kao i za bilo koji drugi model umjetne inteligencije i strojnog učenja, potrebno je trenirati model s velikom bazom podataka. Nakon što se to učini, NLP model će imati prilično dobru predodžbu o vjerojatnosti pojavljivanja iduće riječi nakon određene riječi. Upravo zbog toga što se računalo oslanja na bazu podataka i vjerojatnost pojavljivanja čestih izraza sačinjenih od istih riječi dovodi do toga da i računalo može pogriješiti. Taj nedostatak kod „N-grams“ modela

može se umanjiti korištenjem „N-grams“ većeg stupnja, odnosno korištenjem „trigrams“ ili „4-grams“. Time bi se poboljšao način na koji računalo shvaća kontekst pojavljivanja riječi u rečenicama. Na primjer, korištenjem modela treninga od skupa od 3 povezanih riječi (eng. *trigrams*), virtualni asistent će moći razumjeti razliku između rečenica poput „what is the temperature?“ i „set the temperature“ [22].

U programskoj aplikaciji „N-grams“ se prikupljaju pomoću CountVectorizer funkcije u sklopu sciklit-learn Python biblioteke te uz pomoć bag_of_words modela koji pretvara riječi u numeričke vrijednosti. Da bi se izbjegle riječi poput zamjenica, veznika, priloga, prijedloga i sličnih riječi, koje ne dodaju značaj sadržaju nekog teksta, koristi se tehnika uklanjanja tih riječi koje se u kodu nazivaju „stop_words“. Prikaz koda za prikupljanje „N-grams“ riječi potrebne za funkcioniranje ove tehnike u programskoj aplikaciji prikazan je na slici [Slika 14.].

```
#Get top words
@st.cache
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer(stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

@st.cache
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

@st.cache
def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

Slika 14. Prikaz dijela koda za prikupljanje „N-grams“ riječi

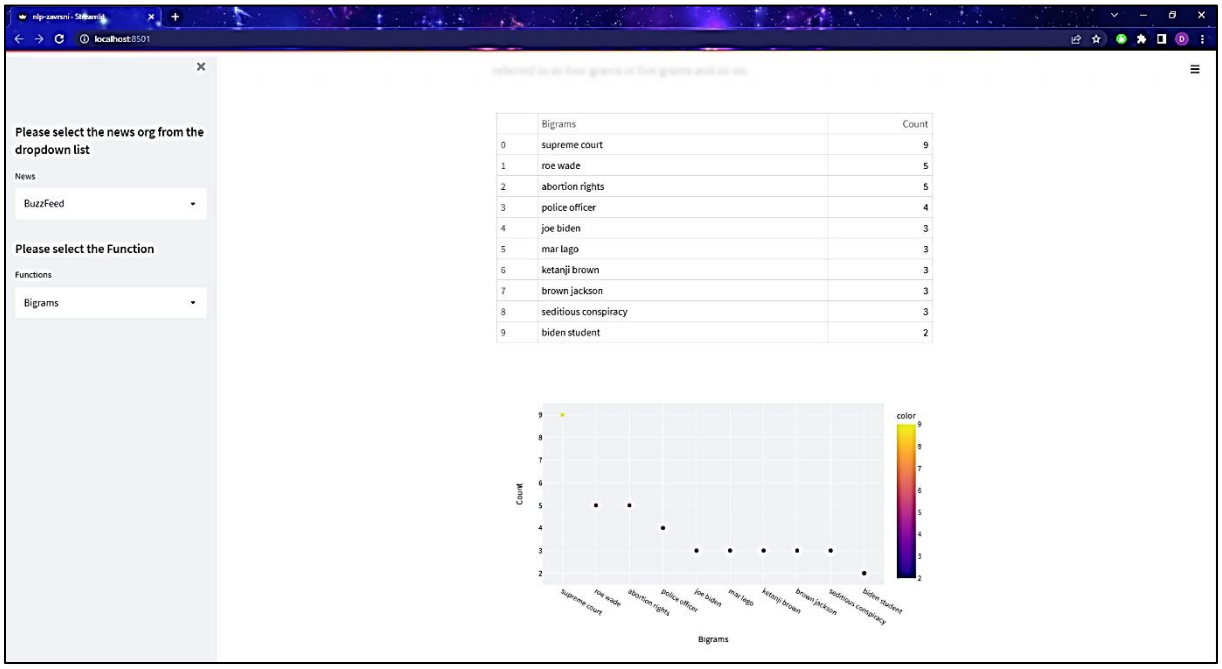
U programskoj aplikaciji moguće je odabrati prikaz „unigrams“, „bigrams“ i „trigrams“ funkcija. Na slici [slika 15.] prikazan je dio koda koji predstavlja slučaj kada se na padajućem izborniku odabere opcija „Bigrams“ te se poziva funkcija koja ispisuje u tablici sve „bigrams“ riječi i graf koji prikazuje njihovu učestalost pojavljivanja na web stranici određenog novinskog portala.

```
if nlp == "Bigrams":
    st.markdown("""
                                                <style>
                                                .bigl-font {
                                                    font-size:20px !important;
                                                }
                                                </style>
                                                """, unsafe_allow_html=True)
    st.write(' ')
    st.markdown('<p class="bigl-font">N Grams</p>', unsafe_allow_html=True)
    st.write(' ')
    st.markdown(
        '<p class="bigl-font">N-grams of texts are extensively used in text mining and natu
        unsafe_allow_html=True)
    st.write(' ')
    st.write(' ')
    common_words = get_top_n_bigram(df_n['title'], 10)
    df4 = pd.DataFrame(common_words, columns=['Bigrams', 'Count'])
    st.table(df4)
    fig = px.scatter(
        x=df4["Bigrams"],
        y=df4["Count"],
        color=df4["Count"],
    )
    fig.update_layout(
        xaxis_title="Bigrams",
        yaxis_title="Count",
    )

    # st.write(fig)
    st.plotly_chart(fig)
# wordcloud.to_image()
```

Slika 15. Prikaz koda za pozivanje „Bigrams“ funkcije

Na slici [Slika 16.] Prikazan je izgled izvođenja „Bigrams“ funkcije u programskoj aplikaciji.



Slika 16. Prikaz „Bigrams“ funkcije u programskoj aplikaciji

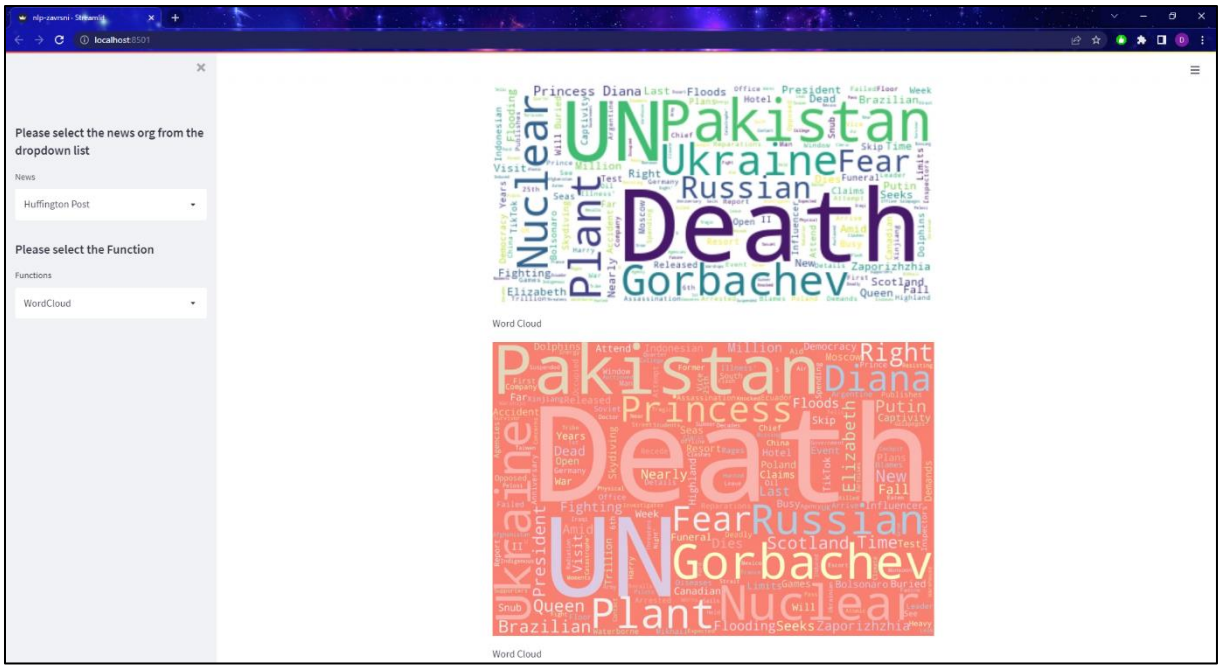
3.3.3 WordCloud

Oblak riječi (eng. *Word Cloud*) je NLP tehnika vizualizacije podataka koja se koristi za predstavljanje tekstualnih podataka u obliku slika. Računalo generira slike sačinjene od često korištenih riječi gdje veličina svake riječi označava njezinu učestalost ili važnost. Tehnika „WordCloud“ često se koristi za analizu podataka sa web stranica društvenih mreža pokušavajući pomoću nje izvući povratne informacije kupaca, zaposlenika ili korisnika. Za generiranja oblaka riječi u Pythonu, potrebni su matplotlib, pandas i wordcloud biblioteke [23]. Na temelju ranije preuzetih tekstualnih informacija pomoću pandas biblioteke, odabiru se naslovi svih članaka koji se pojavljuju na web stranici novinskih portala te pomoću wordcloud i matplotlib funkcija stvaraju slike sačinjene od riječi iz naslova koje se najčešće pojavljuju. Na slici [Slika 17.] prikazan je dio koda koji poziva sve potrebne funkcije iz ranije implementiranih biblioteka za stvaranje oblaka riječi u slučaju da se na padajućem izborniku odabere „WordCloud“ tehnika.

```
if nlp == "WordCloud":
    st.markdown("""
                                                                    <style>
                                                                    .big1-font {
                                                                        font-size:20px !important;
                                                                    }
                                                                    </style>
                                                                    """, unsafe_allow_html=True)
    st.write(' ')
    st.markdown('<p class="big1-font">WordCloud</p>', unsafe_allow_html=True)
    st.write(' ')
    st.markdown(
        '<p class="big1-font">Word clouds or tag clouds are graphical representations of word frequency that give g
        unsafe_allow_html=True)
    st.write(' ')
    st.write(' ')
    long_string = ','.join(list(X_train1.values))
    # Create a WordCloud object
    wordcloud = WordCloud(background_color="white", max_words=5000, contour_width=3, contour_color='steelblue')
    # Generate a word cloud
    wordcloud.generate(long_string)
    # Visualize the word cloud
    plt.figure(figsize=(20, 10))
    plt.imshow(wordcloud)
    st.image(wordcloud.to_array(), width=700)
    st.write("Word Cloud")
    # Generate word cloud
    long_string = ','.join(list(X_train1.values))
    wordcloud = WordCloud(width=3000, height=2000, random_state=1, background_color='salmon', colormap='Pastel1',
                          collocations=False, stopwords=STOPWORDS).generate(long_string)
    # Visualize the word cloud
    plt.figure(figsize=(20, 10))
    plt.imshow(wordcloud)
    st.image(wordcloud.to_array(), width=700)
    st.write("Word Cloud")
    wordcloud = WordCloud(width=3000, height=2000, random_state=1, background_color='black', colormap='Set2',
                          collocations=False, stopwords=STOPWORDS).generate(long_string)
    # Visualize the word cloud
    plt.figure(figsize=(20, 10))
    plt.imshow(wordcloud)
    st.image(wordcloud.to_array(), width=700)
```

Slika 17. Prikaz koda za izvođenje „WordCloud“ tehnike

Na slici [Slika 18.] prikazan je izgled slika koje su nastale generiranjem riječi koje su se najviše pojavljivale na naslovnicama članaka na web stranici novinskog portala Huffington Post 04.09.2022..



Slika 18. Prikaz izvedbe WordCloud tehnike u programskoj aplikaciji

3.3.4 Sentiment Analysis

Računalna analiza sentimenta (eng. *Sentiment analysis*) je proces prepoznavanja pozitivnog ili negativnog sentimenta u nekom tekstu. Ova NLP tehnika često je korištena od strane tvrtki kojima je cilj otkrivanje raspoloženja korisnika njihovih usluga ili razumijevanja želja njihovih kupaca. Računalna analiza sentimenta često je korištena kod:

- Povratnih informacija korisnika,
- Analiziranja društvenih mreža i
- Istraživanja tržišta [24].

Nedostaci ove NLP tehnike najviše se prepoznaju prilikom korištenja ironije ili sarkazma u izražavanju emocija i stavova ljudi. Računalno može neke sarkastične negativne komentare shvatiti kao da su pozitivni zbog nekih uobičajenih izraza koji se uobičajeno povezuju sa pozitivnim sentimentom. Još jedan od problem s ovom tehnikom pojavljuje se kod korištenja različitih vrsta negacija i duple negacije koje mogu dovesti računalo do pogrešnog zaključka o polarnosti određene rečenice. Višeznačnost nekih riječi također može stvarati poteškoće prilikom korištenja ove tehnike [25].

U programskoj aplikaciji razvijenoj u sklopu ovog rada, analiza sentimenta provedena je pomoću TextBlob biblioteke za obradu tekstualnih podataka u Pythonu. Korištenjem ove biblioteke dobivamo numeričke vrijednosti za polarnost rečenica, gdje numerička vrijednost -1 označava negativni, vrijednost jednaka 0 neutralni te vrijednost 1 pozitivan sentiment. Funkcija .sentiment.polarity u sklopu TextBlob biblioteke računa polarnost naslova svih članaka koji se nalaze na web stranici novinskih portala. Na slici [Slika 19.] prikazan je dio koda u kojem se određuje polarnost naslova članaka.

```
@st.cache
def sentiment_textblob(text):
    x = TextBlob(text).sentiment.polarity

    if x < 0:
        return 'neg'
    elif x == 0:
        return 'neu'
    else:
        return 'pos'

#@st.cache(suppress_st_warning=True)
def plot_sentiment_barchart(text, method='TextBlob'):
    if method == 'TextBlob':
        sentiment = text.map(lambda x: sentiment_textblob(x))

    plt.bar(sentiment.value_counts().index,
            sentiment.value_counts(),color=['cyan', 'red', 'green', 'black'],edgecolor='yellow')
    st.set_option('deprecation.showPyplotGlobalUse', False)
    st.pyplot()
```

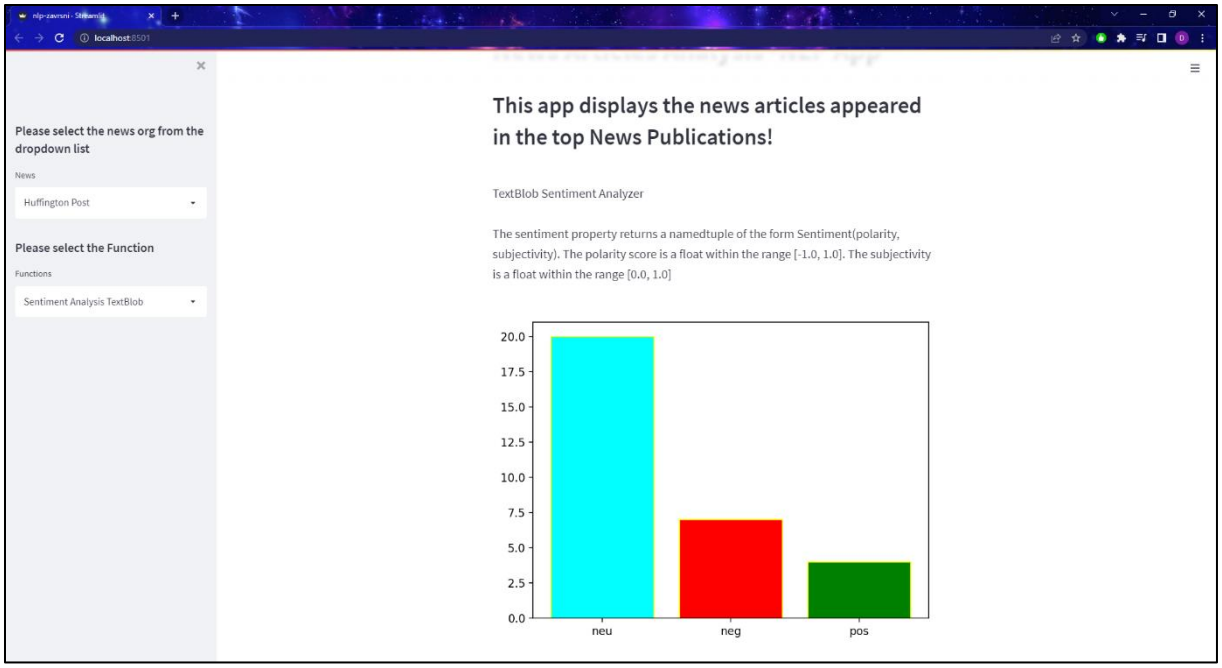
Slika 19. Prikaz koda za određivanje polarnosti u sklopu „Sentiment Analysis“ tehnike

Na slici [Slika 20.] prikazan dio koda koji predstavlja slučaj kada se na padajućem izborniku odabere opcija „Sentiment Analysis TextBlob“ te se poziva funkcija koja izrađuje graf na temelju rezultata izračunate polarnosti.

```
if nlp =="Sentiment Analysis TextBlob":
    st.markdown("""
                        <style>
                        .bigl-font {
                            font-size:20px !important;
                        }
                        </style>
                        """, unsafe_allow_html=True)
    t_word = "The sentiment property returns a namedtuple of the form Sentiment(polarity, subjec
    st.write(' ')
    st.markdown('<p class="bigl-font">TextBlob Sentiment Analyzer</p>',unsafe_allow_html=True)
    st.write(' ')
    st.markdown('<p class="bigl-font">The sentiment property returns a namedtuple of the form Se
    st.write(' ')
    st.write(' ')
    plot_sentiment_barchart(df['title'], method='TextBlob')
```

Slika 20. Prikaz koda za pozivanje „Sentiment Analysis TextBlob“ funkcije

Na slici [Slika 21.] prikazana je izvedba „Sentiment Analysis“ funkcije u programskoj aplikaciji odnosno graf na kojem se prikazuje količina neutralnih „neu“, negativnih „neg“ i pozitivnih „pos“ sentimenata.



Slika 21. Prikaz „Sentiment Analysis“ u programskoj aplikaciji

3.3.5 Parts of Speech Tagging

Vrste riječi u rečenici (eng. *Parts of speech*) objašnjavaju način na koji se riječi koriste u rečenicama. U engleskom jeziku postoje 8 vrsta riječi, a to su imenice, zamjenice, glagoli, pridjevi, prilozi, prijedlozi, veznici i usklici. „Parts of speech“ ili POS tehnika u sklopu NLP-a predstavlja proces pretvaranja rečenica u liste riječi ovisno o njihovoj ulozi u rečenici. Proces odvajanja teksta u „tokene“ (riječi, razmaci, interpunkcije), obavlja se pomoću postupka tokenizacije i word_tokenize funkcije koji su opisani ranije u ovome radu. Nakon postupka tokenizacije, izdvojene riječi se dijele u liste ovisno o njihovoj vrsti. Podjela riječi ovisno o njihovoj vrsti odrađuje se pomoću NLTK biblioteke te funkcije nltk.pos_tag [26]. Na slici [Slika 22.] prikazan je kod koji obavlja proces tokenizacije i podjele riječi ovisno o vrsti.

```
#Parts of Speech Tagging
def plot_parts_of_speach_barchart(text):
    nltk.download('averaged_perceptron_tagger')

    def _get_pos(text):
        pos = nltk.pos_tag(word_tokenize(text))
        pos = list(map(list, zip(*pos)))[1]
        return pos

    tags = text.apply(lambda x: _get_pos(x))
    tags = [x for l in tags for x in l]
    counter = Counter(tags)
    x, y = list(map(list, zip(*counter.most_common(7))))

    sns.barplot(x=y, y=x)
    st.set_option('deprecation.showPyplotGlobalUse', False)
    st.pyplot()
```

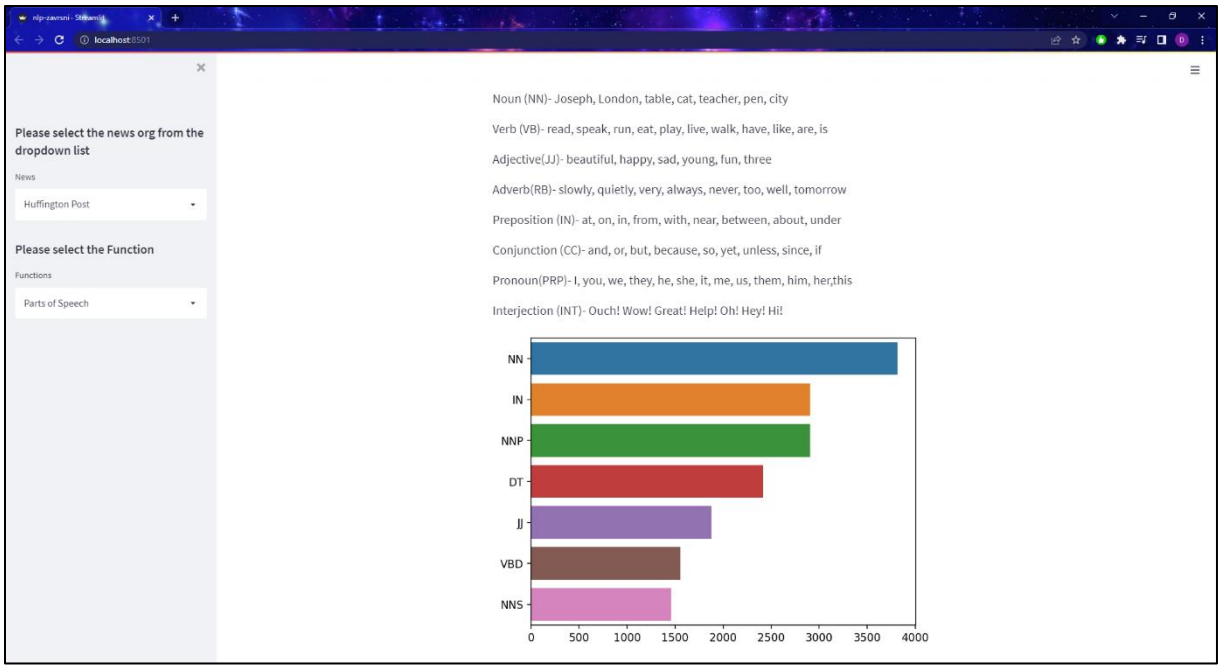
Slika 22. Prikaz dijela koda koji služi za tokenizaciju i podjelu riječi po vrsti

Na slici [Slika 23.] prikazan je dio koda koji predstavlja slučaj kada se na padajućem izborniku odabere opcija „Parts of Speech“.

```
if nlp == "Parts of Speech":
    st.markdown("""
                <style>
                .bigl-font {
                    font-size:20px !important;
                }
                </style>
                """, unsafe_allow_html=True)
    st.write(" ")
    st.markdown('<p class="bigl-font">Noun (NN)- Joseph, London, table, cat, teacher, pen, city</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Verb (VB)- read, speak, run, eat, play, live, walk, have, like, are, is</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Adjective(JJ)- beautiful, happy, sad, young, fun, three</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Adverb(RB)- slowly, quietly, very, always, never, too, well, tomorrow</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Preposition (IN)- at, on, in, from, with, near, between, about, under</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Conjunction (CC)- and, or, but, because, so, yet, unless, since, if</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Pronoun(PRP)- I, you, we, they, he, she, it, me, us, them, him, her,this</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Interjection (INT)- Ouch! Wow! Great! Help! Oh! Hey! Hi!</p>',unsafe_allow_html=True)
    plot_parts_of_speach_barchart(df['content'])
```

Slika 23. Prikaz koda za odabir „Parts of Speech“ tehnike

Na slici [Slika 24.] Prikazana je izvedba „Parts of Speech“ tehnike u programskoj aplikaciji. Na vrhu stranice program ispisuje na koje se sve vrste dijele riječi u rečenici te neki primjeri istih. Na dnu stranice nalazi se graf koji pokazuje količinu vrsti riječi u rečenici koje se pojavljuju u člancima određenih novinskih portala.



Slika 24. prikaz „Parts of Speech Tagging“ u programskoj aplikaciji

4. ZAKLJUČAK

U radu je prikazan razvoj programske aplikacije za analizu članaka sa web stranica novinskih portala. Teorijska osnova te razvoj računalne analize i prepoznavanja govora obrađeni su na početku rada. Poznavanjem teorijskih osnova omogućeno je shvaćanje načina na koji programska aplikacija funkcionira. U razvijenoj aplikaciji, analiza članaka vrši se pomoću NLP tehnika, koje imaju svoje prednosti i nedostatke. Nedostatci kod korištenja ovih tehnika najlakše se mogu prepoznati kod „N-grams“ i „Sentiment Analysis“ tehnika. Kod primjene „N-grams“ tehnike problem nastupa ako se obrađuje tekst sa prepoznavanjem malog broja povezanih riječi u rečenici „unigrams“ ili „bigrams“ te računalo ne može dovoljno dobro povezati kontekst u kojem se ta riječ ponavlja i pojavljuje. Rješenje tog problema se izbjegava korištenjem većeg broja povezanih riječi unutar rečenica poput „trigrams“ i „4-grams“. Prilikom uporabe „Sentiment Analysis“ tehnike najveći problem predstavlja shvaćanje sarkazma ili ironije koju korisnici često koriste te računalo nije u mogućnosti raspoznati je li neki tekst pozitivnog ili negativnog karaktera. Takvi nedostatci bi se mogli u budućnosti riješiti daljnjim razvijanjem NLP biblioteka i učenja softvera o takvom načinu govora. Tokom izrade ovog rada, korištenjem programske aplikacije kroz dulji vremenski period, moglo se uočiti da u različitim vremenskim periodima ovisno o aktualnostima u svijetu i vijestima koje se prikazuju, dolazi do značajnih promjena u prikazu rezultata NLP tehnika. To opažanje najviše se može primijetiti primjenom „WordCloud“ tehnike koja na slikoviti način prikazuje što je trenutno najaktualnija tema u svijetu.

LITERATURA

[1] https://en.wikipedia.org/wiki/Natural_language_processing#Neural_NLP_(present) dostupno dana 26.08.2022.

[2] https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML dostupno dana 26.08.2022.

[3] https://hci.stanford.edu/~winograd/shrdlu/ dostupno dana 26.08.2022.

[4] https://pcchip.hr/ostalo/tech/strojno-ucenje-vs-duboko-ucenje/ dostupno dana 27.08.2022.

[5] https://en.wikipedia.org/wiki/Neural_network dostupno dana 27.08.2022.

[6] https://en.wikipedia.org/wiki/Statistical_model dostupno dana 27.08.2022.

[7] https://en.wikipedia.org/wiki/Speech_recognition dostupno dana 28.08.2022.

[8] https://proxet.com/blog/fundamentals-of-statistical-natural-language-processing/ dostupno dana 28.08.2022.

[9] https://hr.wikipedia.org/wiki/Umjetna_neuronska_mre%C5%BEa dostupno dana 28.08. 2022.

[10] https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks dostupno dana 29.08.2022.

[11] https://mspoweruser.com/bizarre-study-claims-cortana-has-19-digital-assistant-market-share/ dostupno dana 29.08.2022.

[12] https://hr.wikipedia.org/wiki/Python_(programski_jezik) dostupno dana 29.08.2022.

[13] https://en.wikipedia.org/wiki/Pandas_(software) dostupno dana 30.08.2022.

[14] https://dev.to/mr_destructive/feedparser-python-package-for-reading-rss-feeds-5fnc dostupno dana 30.08.2022.

[15] https://www.w3schools.com/python/numpy/numpy_intro.asp dostupno dana 30.08.2022.

[16] https://matplotlib.org/ dostupno dana 31.08.2022.

[17] https://www.nltk.org/ dostupno dana 31.08.2022.

[18] https://www.geeksforgeeks.org/python-lemmatization-with-nltk/ dostupno dana 31.08. 2022.

[19] https://www.educative.io/answers/what-is-wordtokenize-in-python dostupno dana 01.09. 2022.

[20] https://realpython.com/natural-language-processing-spacy-python/ dostupno dana 01.09. 2022.

[21] https://analyticsindiamag.com/beautiful-soup-webscraping-python/ dostupno dana 01. 09.2022.

[22] https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058 dostupno dana 02.09.2022.

[23] https://www.geeksforgeeks.org/generating-word-cloud-python/ dostupno dana 02.09. 2022.

[24] https://en.wikipedia.org/wiki/Sentiment_analysis dostupno dana 02.09.2022.

[25] https://www.toptal.com/deep-learning/4-sentiment-analysis-accuracy-traps dostupno dana 03.09.2022.

[26] https://pythonalgos.com/natural-language-processing-part-of-speech-tagging/ dostupno dana 03.09.2022.

[27] https://developer.nvidia.com/blog/how-to-build-domain-specific-automatic-speech-recognition-models-on-gpus/ dostupno dana 04.09.2022.

[28] https://www.turing.com/kb/stemming-vs-lemmatization-in-python dostupno dana 04.09. 2022.

PRILOZI

I. Python kod (dio koda za jedan od četiri novinskih portala)

```
#Import all the required livraries..

import pandas as pd
import feedparser
# Import packages
import matplotlib.pyplot as plt
#get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import gensim
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import word_tokenize
import pyLDAvis.gensim_models
from collections import Counter
import feedparser
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import streamlit as st
import base64
import seaborn as sns
from PIL import Image
import cufflinks
from sklearn.feature_extraction.text import CountVectorizer
from plotly.offline import iplot
from textblob import TextBlob
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import plotly.express as px
import spacy
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.manifold import TSNE
#from gensim.summarization import summarize
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.lex_rank import LexRankSummarizer
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.lsa import LsaSummarizer as Summarizer
from sumy.utils import get_stop_words
from sumy.nlp.stemmers import Stemmer
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer as sumytoken
from sumy.summarizers.luhn import LuhnSummarizer

nltk.download('punkt')
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')
```

```
# In[ ]:
class RSSFeed():
    feedurl = ""

    global ndf
    def __init__(self, paramrssurl):
        print(paramrssurl)
        self.feedurl = paramrssurl
        self.parse()


    def parse(self):
        thefeed = feedparser.parse(self.feedurl)
        global ndf
        ndf = pd.DataFrame(columns=['title', 'link', 'decription', 'content'])
        for thefeedentry in thefeed.entries:
            title = thefeedentry.get("title", "")
            link = thefeedentry.get("link", "")
            decr = thefeedentry.get("description", "")
            ndf = ndf.append({'title': title, 'link': link, 'decription': decr},
                             ignore_index=True)
        return ndf

#Beautiful Soup Code
@st.cache
def full_text(my_url):
    import requests
    from bs4 import BeautifulSoup
    import pandas as pd
    url = my_url
    article = requests.get(url)
    articles = BeautifulSoup(article.content, 'html.parser')
    articles_body = articles.findAll('body')
    p_blocks = articles_body[0].findAll('p')
    p_blocks_df = pd.DataFrame(columns=['element_name', 'parent_hierarchy', 'element_text', 'element_text_Count'])
    for i in range(0, len(p_blocks)):
        parents_list = []
        for parent in p_blocks[i].parents:
            Parent_id = ''
            try:
                Parent_id = parent['id']
            except:
                pass
            parents_list.append(parent.name + 'id: ' + Parent_id)
        parent_element_list = ['' if (x == 'None' or x is None) else x for x in parents_list]
        parent_element_list.reverse()
        parent_hierarchy = ' -> '.join(parent_element_list)
        p_blocks_df = p_blocks_df.append({"element_name": p_blocks[i].name
                                          , "parent_hierarchy": parent_hierarchy
                                          , "element_text": p_blocks[i].text
                                          , "element_text_Count": len(str(p_blocks[i].text))}
                                         , ignore_index=True
                                         , sort=False)
    if len(p_blocks_df) > 0:
        p_blocks_df_groupby_parent_hierarchy = p_blocks_df.groupby(by=['parent_hierarchy'])
        p_blocks_df_groupby_parent_hierarchy_sum = p_blocks_df_groupby_parent_hierarchy[['element_text_Count']].sum()
        p_blocks_df_groupby_parent_hierarchy_sum.reset_index(inplace=True)
    maxid = p_blocks_df_groupby_parent_hierarchy_sum.loc[
        p_blocks_df_groupby_parent_hierarchy_sum['element_text_Count'].idxmax()
```

```
        p_blocks_df_groupby_parent_hierarchy_sum['element_text_Count'].idxmax()
        , 'parent_hierarchy']
    merge_text = '\n'.join(p_blocks_df.loc[p_blocks_df['parent_hierarchy'] == maxid, 'element_text'].to_list())
    return merge_text


#Matplt lib
@st.cache
def preprocess(ReviewText):
    ReviewText = ReviewText.str.replace("(<br/>)", "")
    ReviewText = ReviewText.str.replace('(<a).*(>).*(</a>)', '')
    ReviewText = ReviewText.str.replace('(&amp)', '')
    ReviewText = ReviewText.str.replace('(&gt)', '')
    ReviewText = ReviewText.str.replace('(&lt)', '')
    ReviewText = ReviewText.str.replace('(\xa0)', ' ')
    return ReviewText


#Get top words
@st.cache
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer(stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

@st.cache
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

@st.cache
def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

@st.cache
def sentiment_textblob(text):
    x = TextBlob(text).sentiment.polarity

    if x < 0:
        return 'neg'
    elif x == 0:
        return 'neu'
    else:
        return 'pos'

#@st.cache(suppress_st_warning=True)
def plot_sentiment_barchart(text, method='TextBlob'):
    if method == 'TextBlob':
        sentiment = text.map(lambda x: sentiment_textblob(x))
```

```
        sentiment = text.map(lambda x: sentiment_textblob(x))

    plt.bar(sentiment.value_counts().index,
            sentiment.value_counts(),color=['cyan', 'red', 'green', 'black'],edgecolor='yellow')
    st.set_option('deprecation.showPyplotGlobalUse', False)
    st.pyplot()

#Parts of Speech Tagging
def plot_parts_of_speach_barchart(text):
    nltk.download('averaged_perceptron_tagger')

    def _get_pos(text):
        pos = nltk.pos_tag(word_tokenize(text))
        pos = list(map(list, zip(*pos)))[1]
        return pos

    tags = text.apply(lambda x: _get_pos(x))
    tags = [x for l in tags for x in l]
    counter = Counter(tags)
    x, y = list(map(list, zip(*counter.most_common(7))))

    sns.barplot(x=y, y=x)
    st.set_option('deprecation.showPyplotGlobalUse', False)
    st.pyplot()

#st.set_page_config(layout="wide")
st.title('News Articles Analysis -NLP App')
st.header("""
This app displays the news articles appeared in the top News Publications!
""")

st.sidebar.header('Please select the news org from the dropdown list')
lnews = [ "NY Times","BuzzFeed","Huffington Post","The Wall Street Journal"]
s_news = st.sidebar.selectbox('News', lnews)
st.sidebar.header('Please select the Function')
lnlp = ["Intro","Snapshot","Unigrams","Bigrams","Trigrams","WordCloud","Sentiment Analysis TextBl
s_nlp = st.sidebar.selectbox('Functions', lnlp)

def load_data(news,nlp):
   if news =="NY Times":
       #st.write(news)
       #st.write(nlp)
       if nlp =="Intro":
           #st.write("this is intro")
           image1 = Image.open(r'C:\Users\Domagoj\Desktop\FSB\1. PREDDIPLOMSKI STUDIJ\4. godina z
           st.write( " ")
           st.image(image1, width=300)
           st.write(" ")
           st.write(" ")
           st.write(" ")
           image = Image.open(r'C:\Users\Domagoj\Desktop\FSB\1. PREDDIPLOMSKI STUDIJ\4. godina zi
           st.image(image, width=700)
           st.write(" ")
           st.write(" ")
           st.write(" ")
           st.markdown("""
           The New York Times (NYT or NY Times) is an American daily newspaper based in New York
           The paper is owned by The New York Times Company, which is publicly traded. It has bee
           Since the mid-1970s, The New York Times has expanded its layout and organization, addi
           The Times stayed with the broadsheet full-page set-up and an eight-column format for s
           """)
```

```
    """)

df = pd.DataFrame(columns=['title', 'link', 'decription', 'content'])
url_link = "https://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml"
RSSFeed(url_link)
df = ndf
#st.header('Display the dataframe')
#st.dataframe(df)
pd.set_option('display.max_rows', df.shape[0] + 1)
df.reset_index(inplace=True, drop=True)
for ind in df.index:
    # print(df['title'][ind], df['link'][ind], df['content'][ind])
    url = df['link'][ind]
    #print(url)
    text = full_text(url)
    df['content'][ind] = text
#st.write(df['title'])
# Build the corpus.
corpus = []
for ind in df.index:
    # corpus = df['content'][ind]
    corpus.append(df['title'][ind])
#print(corpus)
df = df.dropna()
X_train1 = df['title']
if nlp == "Snapshot":
    st.write(" ")
    st.write(" ")
    st.write(" ")
    st.subheader('Display the dataframe')
    st.write(" ")
    st.write(" ")
    st.write(" ")
    st.dataframe(df)
    st.write(" ")
    st.write(" ")
    st.write(" ")
    st.markdown("""
                                                                    <style>
                                                                    .big2-font {
                                                                        font-size:30px !important;
                                                                    }
                                                                    </style>
                                                                    """, unsafe_allow_html=True)
    st.markdown('<p class="big2-font">The no of articles :</p>', unsafe_allow_html=True)
    st.write(df.shape[0])
    st.write(" ")
    st.write("The Url Link ")
    for index, row in df.iterrows():
        st.write(row['link'])

if nlp == "WordCloud":
    st.markdown("""
                                                                <style>
                                                                .big1-font {
                                                                    font-size:20px !important;
                                                                }
                                                                </style>
                                                                """, unsafe_allow_html=True)
    st.write(' ')
    st.markdown('<p class="big1-font">WordCloud</p>', unsafe_allow_html=True)
```

```
        st.markdown('<p class="bigl-font">WordCloud</p>', unsafe_allow_html=True)
        st.write(' ')
        st.markdown(
            '<p class="bigl-font">Word clouds or tag clouds are graphical representations of word frequency that give gr
            unsafe_allow_html=True)
        st.write(' ')
        st.write(' ')
        long_string = ','.join(list(X_train1.values))
        # Create a WordCloud object
        wordcloud = WordCloud(background_color="white", max_words=5000, contour_width=3, contour_color='steelblue')
        # Generate a word cloud
        wordcloud.generate(long_string)
        # Visualize the word cloud
        plt.figure(figsize=(20, 10))
        plt.imshow(wordcloud)
        st.image(wordcloud.to_array(), width=700)
        st.write("Word Cloud")
        # Generate word cloud
        long_string = ','.join(list(X_train1.values))
        wordcloud = WordCloud(width=3000, height=2000, random_state=1, background_color='salmon', colormap='Pastell',
                              collocations=False, stopwords=STOPWORDS).generate(long_string)
        # Visualize the word cloud
        plt.figure(figsize=(20, 10))
        plt.imshow(wordcloud)
        st.image(wordcloud.to_array(), width=700)
        st.write("Word Cloud")
        wordcloud = WordCloud(width=3000, height=2000, random_state=1, background_color='black', colormap='Set2',
                              collocations=False, stopwords=STOPWORDS).generate(long_string)
        # Visualize the word cloud
        plt.figure(figsize=(20, 10))
        plt.imshow(wordcloud)
        st.image(wordcloud.to_array(), width=700)

    df_n = df
    df_n['title'] = preprocess(df['title'])

    if nlp == "Unigrams":
        st.markdown("""
                                                <style>
                                                .bigl-font {
                                                    font-size:20px !important;
                                                }
                                                </style>
                                                """, unsafe_allow_html=True)
        st.write(' ')
        st.markdown('<p class="bigl-font">N Grams</p>', unsafe_allow_html=True)
        st.write(' ')
        st.markdown(
            '<p class="bigl-font">N-grams of texts are extensively used in text mining and natural language processing t
            unsafe_allow_html=True)
        st.write(' ')
        st.write(' ')

        common_words = get_top_n_words(df_n['title'], 10)
        #for word, freq in common_words:
        #    (word, freq)
        df2 = pd.DataFrame(common_words, columns=['Words', 'Count'])
        st.table(df2)
        fig = px.scatter(
            x=df2["Words"],
            y=df2["Count"],
```

```
            y=df2["Count"],
            color=df2["Count"],
        )
        fig.update_layout(
            xaxis_title="Words",
            yaxis_title="Count",
        )

        # st.write(fig)
        st.plotly_chart(fig)

    if nlp == "Bigrams":
        st.markdown("""
                                                        <style>
                                                        .bigl-font {
                                                            font-size:20px !important;
                                                        }
                                                        </style>
                                                        """, unsafe_allow_html=True)
        st.write(' ')
        st.markdown('<p class="bigl-font">N Grams</p>', unsafe_allow_html=True)
        st.write(' ')
        st.markdown(
            '<p class="bigl-font">N-grams of texts are extensively used in text mining and nat
            unsafe_allow_html=True)
        st.write(' ')
        st.write(' ')
        common_words = get_top_n_bigram(df_n['title'], 10)
        df4 = pd.DataFrame(common_words, columns=['Bigrams', 'Count'])
        st.table(df4)
        fig = px.scatter(
            x=df4["Bigrams"],
            y=df4["Count"],
            color=df4["Count"],
        )
        fig.update_layout(
            xaxis_title="Bigrams",
            yaxis_title="Count",
        )

        # st.write(fig)
        st.plotly_chart(fig)
    # wordcloud.to_image()

    if nlp == 'Trigrams':
        st.markdown("""
                                                        <style>
                                                        .bigl-font {
                                                            font-size:20px !important;
                                                        }
                                                        </style>
                                                        """, unsafe_allow_html=True)
        st.write(' ')
        st.markdown('<p class="bigl-font">N Grams</p>', unsafe_allow_html=True)
        st.write(' ')
        st.markdown(
            '<p class="bigl-font">N-grams of texts are extensively used in text mining and nat
            unsafe_allow_html=True)
        st.write(' ')
        st.write(' ')
        common_words = get_top_n_trigram(df_n['title'], 10)
```

```
    common_words = get_top_n_trigram(df_n['title'], 10)
    df6 = pd.DataFrame(common_words, columns=['Trigrams', 'Count'])
    st.table(df6)
    fig = px.scatter(
        x=df6["Trigrams"],
        y=df6["Count"],
        color=df6["Count"],
    )
    fig.update_layout(
        xaxis_title="Trigrams",
        yaxis_title="Count",
    )

    # st.write(fig)
    st.plotly_chart(fig)

if nlp =="Sentiment Analysis TextBlob":
    st.markdown("""
                        <style>
                        .bigl-font {
                            font-size:20px !important;
                        }
                        </style>
                        """, unsafe_allow_html=True)
    t_word = "The sentiment property returns a namedtuple of the form Sentiment(polarity, subjectivity). The polarity score is a float withi
    st.write(' ')
    st.markdown('<p class="bigl-font">TextBlob Sentiment Analyzer</p>',unsafe_allow_html=True)
    st.write(' ')
    st.markdown('<p class="bigl-font">The sentiment property returns a namedtuple of the form Sentiment(polarity, subjectivity). The polarit
    st.write(' ')
    st.write(' ')
    plot_sentiment_barchart(df['title'], method='TextBlob')

if nlp == "Parts of Speech":
    st.markdown("""
               <style>
               .bigl-font {
                   font-size:20px !important;
               }
               </style>
               """, unsafe_allow_html=True)
    st.write(" ")
    st.markdown('<p class="bigl-font">Noun (NN)- Joseph, London, table, cat, teacher, pen, city</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Verb (VB)- read, speak, run, eat, play, live, walk, have, like, are, is</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Adjective(JJ)- beautiful, happy, sad, young, fun, three</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Adverb(RB)- slowly, quietly, very, always, never, too, well, tomorrow</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Preposition (IN)- at, on, in, from, with, near, between, about, under</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Conjunction (CC)- and, or, but, because, so, yet, unless, since, if</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Pronoun(PRP)- I, you, we, they, he, she, it, me, us, them, him, her,this</p>',unsafe_allow_html=True)
    st.markdown('<p class="bigl-font">Interjection (INT)- Ouch! Wow! Great! Help! Oh! Hey! Hi!</p>',unsafe_allow_html=True)
    plot_parts_of_speach_barchart(df['content'])
```