Algoritmi za određivanje sličnosti teksta i njihova implementacija u modelu web aplikacije

Poljak, Josip

Sveučilište
Sjever

Završni rad br. 9/RINF/2024

Algoritmi za određivanje sličnosti teksta i njihova implementacija u modelu web aplikacije

Josip Poljak, 0016084831

Đurđevac, rujan 2024. godine

Sveučilište
Sjever

Odjel za računarstvo i informatiku

Završni rad br. 9/RINF/2024

Algoritmi za određivanje sličnosti teksta i njihova implementacija u modelu web aplikacije

Student

Josip Poljak, 0016084831

Mentor

Dražen Crčić, mag.ing.el.

Đurđevac, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Koprivnica
Trg dr. Žarka Dolinara 1, HR-48000 Koprivnica

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Računarstvo i informatika

STUDIJ: Stručni prijediplomski studij Računarstvo i informatika

PRISTUPNIK: Josip Poljak | MATIČNI BROJ: 0016084831

DATUM: 26. 8. 2024. | KOLEGIJ: Računala u poslovnoj primjeni

NASLOV RADA: Algoritmi za određivanje sličnosti teksta i njihova implementacija u modelu web aplikacije

NASLOV RADA NA ENGL. JEZIKU: Algorithms for determining text similarity and their implementation in a web application model

MENTOR: Dražen Crčić | ZVANJE: predavač

ČLANOVI POVJERENSTVA:
1. doc.dr.sc. Tomislav Horvat - predsjednik povjerenstva
2. mr.sc. Vladimir Stanisavljević, v.pred - član povjerenstva
3. pred. Dražen Crčić, mag.ing. - mentor
4. doc.dr.sc. Domagoj Frank - zamjenski član
5.

Zadatak završnog rada

BROJ: 9/RINF/2024

OPIS:

Algoritmi za određivanje sličnosti teksta omogućavaju kvantitativnu i kvalitativnu procjenu sličnosti između tekstova, što je korisno u mnogim područjima poput pretraživanja informacija, sažimanja teksta te detekcije plagijata. Mnogi se koriste između ostalog i u raznim web aplikacijama u konkretnu svrhu.

U sklopu završnog rada potrebno je:
- objasniti osnovne algoritme za određivanje sličnosti teksta
- izraditi vlastitiu web aplikaciju koja koristi navedene algoritme te uspoređuje uneseni tekst
- objasniti korištene tehnologije u izradi rješenja
- testirati konačno rješenje te opisati dobivene rezultate

ZADATAK URUČEN: 27.8.24 | POTPIS MENTORA:

SVEUČILIŠTE SJEVER

Očisti obrazac

Sažetak

U digitalnom dobu, tekst je ključan medij za komunikaciju i prenošenje informacija. S obzirom na ogromne količine tekstualnih podataka koje se stvaraju i konzumiraju svakodnevno, postoji sve veća potreba za alatima koji mogu usporediti, analizirati i interpretirati tekstove. Algoritmi za mjerenje sličnosti teksta omogućavaju kvantitativnu i kvalitativnu procjenu sličnosti između tekstova, što je korisno u mnogim područjima poput pretraživanja informacija, sažimanja teksta, detekcije plagijata i analize sentimenta.

Računarska lingvistika koristi različite algoritme za analizu teksta, gdje svaki algoritam ima svoje specifičnosti, prednosti i nedostatke. Algoritmi poput Levenshteinove i Damerau-Levenshteinove udaljenosti, Jaro i Jaro-Winkler sličnosti, Smith-Watermanove sličnosti, kosinusne sličnosti i N-gram sličnosti razmatrani su detaljno, ističući njihove primjene i efikasnost u različitim kontekstima.

Algoritmi kao što su Levenshteinova i Damerau-Levenshteinova udaljenost su efikasni u detekciji i ispravljanju pravopisnih grešaka, dok Jaro i Jaro-Winkler sličnost služe u identifikaciji sličnosti u kraćim tekstovima. Smith-Watermanova sličnost se ističe u identifikaciji sličnih segmenata unutar dužih tekstova. Kosinusna sličnost je idealna za obradu većih tekstualnih korpusa, a N-gram sličnost je korisna u fonetskim i ortografskim analizama na mikro razini.

Važno je odabrati pravilan algoritam za mjerenje sličnosti teksta u specifičnim kontekstima računarske lingvistike. Svaki algoritam ima svoju ulogu, ovisno o specifičnom zadatku i kontekstu. Postoji potencijal za razvoj hibridnih algoritama koji bi kombinirali prednosti pojedinih metoda. Buduća istraživanja mogla bi istražiti kako napredne tehnologije kao što su umjetna inteligencija i strojno učenje mogu dalje unaprijediti ove algoritme. U konačnici, kombinacija ovih algoritama predstavlja neophodan skup alata za stručnjake u računalnoj lingvistici, ključan za suvremeno razumijevanje i analizu tekstualnih podataka.

Ključne riječi: Algoritmi za sličnost teksta, Levenshtein udaljenost, Damerau-Levenshtein udaljenost, Jaro sličnost, Jaro-Winkler sličnost, Smith-Waterman sličnost, Kosinusna sličnost, N-gram sličnost, računarska lingvistika, obrada prirodnog jezika (NLP)

Summary

In the digital age, text is a crucial medium for communication and information transmission. Given the vast amounts of textual data created and consumed daily, there is a growing need for tools that can compare, analyze, and interpret texts. Algorithms for measuring text similarity enable both quantitative and qualitative assessment of similarity between texts, which is valuable in various fields such as information retrieval, text summarization, plagiarism detection, and sentiment analysis.

Computational linguistics employs different algorithms for text analysis, each with its specific characteristics, advantages, and drawbacks. Algorithms such as Levenshtein and Damerau-Levenshtein distance, Jaro and Jaro-Winkler similarity, Smith-Waterman similarity, cosine similarity, and N-gram similarity have been discussed in detail, highlighting their applications and efficiency in different contexts.

Algorithms like Levenshtein and Damerau-Levenshtein distance are effective in detecting and correcting spelling errors, while Jaro and Jaro-Winkler similarity are useful for identifying similarities in shorter texts. Smith-Waterman similarity excels in identifying similar segments within longer texts. Cosine similarity is ideal for processing larger text corpora, and N-gram similarity is useful in phonetic and orthographic analyses on a micro level.

It is important to choose the right algorithm for measuring text similarity in specific computational linguistics contexts. Each algorithm plays its role depending on the specific task and context. There is potential for the development of hybrid algorithms that combine the strengths of individual methods. Future research could explore how advanced technologies such as artificial intelligence and machine learning can further improve these algorithms. Ultimately, a combination of these algorithms represents an essential set of tools for experts in computational linguistics, crucial for modern understanding and analysis of textual data.

Keywords: Text similarity algorithms, Levenshtein distance, Damerau-Levenshtein distance, Jaro similarity, Jaro-Winkler similarity, Smith-Waterman similarity, Cosine similarity, N-gram similarity, computational linguistics, natural language processing (NLP)

Popis korištenih kratica

NLP - Neurolingvističko programiranje

PHP - Hypertext Preprocessor - programski jezik

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

Psutil - Python system and process utilities

AJAX - Asynchronous JavaScript and XML

DOM - Document Object Model

HTTP - Hypertext Transfer Protocol

JSON - JavaScript Object Notation

PDF - Portable Document Format

Sadržaj

1. Uvod

U današnje vrijeme kada je digitalizacija sveprisutnija, tekst je osnovni način komunikacije i prenošenja informacija među ljudima. Svaki dan, stvaraju se i konzumiraju ogromne količine tekstualnih podataka, bilo u obliku pisanih dokumenata, web stranica, društvenih mreža, e-mailova, poruka i sl. Međutim, nije sav tekst jednak po sadržaju, stilu, kvaliteti i značenju. Česte su situacije u kojima je potrebno usporediti, analizirati, sažeti ili prevesti različite tekstove radi boljeg uvida u njihovu sličnost, razliku, relevantnost ili vrijednost. Ovi zadaci su vrlo važni i korisni u mnogim područjima, kao što su pretraživanje informacija, sažimanje teksta, detekcija plagijata, analiza sentimenta, prepoznavanje entiteta i još mnogo toga.

Za rješavanje ovakvih zadataka, potrebni su algoritmi koji mogu mjeriti sličnost teksta na osnovu različitih kriterija i metoda. Algoritmi za sličnost teksta su skup postupaka i tehnika koje omogućavaju da se kvantitativno i kvalitativno procijeni koliko su dva ili više tekstova slični ili različiti po nekom aspektu, kao što su struktura, sadržaj, stil, značenje i sl. Ovi algoritmi se temelje na različitim pristupima i metodama, kao što su vektorski modeli, latentna semantička analiza, neuronske mreže, semantičke mreže i ontologije. Svaki od ovih pristupa i metoda ima svoje prednosti i nedostatke te se razlikuju po složenosti, preciznosti i efikasnosti.

Mjerenje sličnosti teksta je važan i koristan zadatak u računarskoj lingvistici jer omogućava bolje razumijevanje, pretraživanje, sažimanje, prevođenje i analiziranje tekstova. Na primjer, ako se želi pronaći relevantne informacije o nekoj temi, moguće je koristiti algoritme za sličnost teksta za usporedbu upita sa različitim izvorima i odabir onih koji su najbliži interesu. Ukoliko se želi sažeti dugi tekst u kraći, mogu se koristiti algoritmi za sličnost teksta kako bi se identificirale glavne ideje i ključne riječi u tekstu i izbacili nepotrebni detalji. Prilikom detektiranja plagijata u nekom radu, mogu se koristiti algoritmi za sličnost teksta za usporedbu rada s drugim radovima i otkrivanje sličnih ili identičnih dijelova. Ukoliko je potrebno analizirati sentiment u nekom tekstu, mogu se koristiti algoritmi za sličnost teksta za usporedbu teksta s rječnikom pozitivnih i negativnih riječi i određivanje tona i stava teksta. Prilikom prepoznavanja entiteta u nekom tekstu, mogu se koristiti algoritmi za sličnost teksta za usporedbu teksta s bazom podataka entiteta i izdvajanje imena, lokacije, organizacije i sl.

Međutim, mjerenje sličnosti teksta nije jednostavan i jednoznačan zadatak jer postoje mnogi faktori koji utječu na percepciju i interpretaciju tekstova. Na primjer, tekstovi mogu biti napisani na različitim jezicima, koji imaju različite gramatičke, sintaktičke i semantičke osobine. Tekstovi mogu biti napisani u različitim stilovima, koji imaju različite razine formalnosti, složenosti i kreativnosti. Također, mogu biti napisani u različitim kontekstima,

koji imaju različite ciljeve, publike i poruke. Tekstovi mogu imati različite razine sličnosti, koji mogu biti bazirani na različitim aspektima, kao što su struktura, sadržaj, stil, značenje i sl. Svi ovi faktori otežavaju i kompliciraju mjerenje sličnosti teksta te zahtijevaju različite pristupe i metode za njegovo rješavanje.

U ovom radu, cilj je istražiti, analizirati i usporediti neke od najpoznatijih i najčešće korištenih algoritama za sličnost teksta te pokazati njihove primjene i efikasnost u različitim kontekstima i problemima u području računarske lingvistike.

2. Teorijski pregled

Puno današnjih aplikacija koje uključuju povezivanje sadržaja i ispravke pravopisa, prepoznavanje govora i sekvenciranje gena, oslanjaju se na određene kvantitativne metrike kako bi odredile mjeru sličnosti nizova. Računanje sličnosti nizova može pomoći u rješavanju ovih problema, no općenito je računalno zahtjevno i automatski ne daje idealne rezultate zbog raznolike i nejasne prirode mogućih grešaka u podacima.

S obzirom na to, trebat će odabrati optimalne algoritme za korištenje iz čitavog skupa dostupnih algoritama. U nastavku su detaljno obrađeni algoritmi koji su najkorišteniji kod mjerenja sličnosti tekstova.

2.1 Računarska lingvistika i algoritmi za sličnost teksta

Ovo poglavlje pruža uvid u računarsku lingvistiku koja se bavi razvojem i primjenom računarskih metoda za obradu prirodnog jezika, uključujući morfološku, semantičku i sintaktičku analizu te naglašava važnost algoritama za mjerenje sličnosti teksta ističući ih kao ključne instrumente u analizi i organizaciji informacija.

2.1.1. Osnove i značaj računarske lingvistike

Računarska lingvistika je interdisciplinarno područje koje se bavi razvojem i primjenom računarskih metoda za obradu prirodnog jezika (engl. *natural language processing* – NLP). Ova disciplina proizlazi iz potrebe da se omogući računalima razumijevanje i obrada ljudskog jezika. Ova grana informatike kombinira elemente jezikoslovlja, računalne znanosti te statističkih metoda kako bi stvorila alate i tehnike za rad s prirodnim jezikom. Rezultate objedinjuje iz teoretskog računarstva, jezikoslovlja, kognitivne psihologije, umjetne inteligencije i logike. Fokusira se na razvoj algoritama, modela i tehnika koji omogućavaju računalima rad s prirodnim jezikom, kao što su potpora korisnika računala pri obradbi tekstova, automatsko traženje određenih mjesta u tekstovima, potpora pri prevođenju tekstova iz jednog jezika u drugi, obradba govora, pronalaženje informacija, potpora autoru pri pisanju tekstova, interakcija između čovjeka i računala u prirodnom jeziku i sl.

Ključni aspekti obuhvaćaju morfološku, sintaktičku i semantičku analizu. Morfološka analiza odnosi se na razumijevanje osnovnih građevnih jedinica riječi poput korijena, prefiksa i sufiksa te dekompoziciju riječi na njihove osnovne dijelove. Sintaktička analiza se fokusira na razumijevanje strukture rečenica i međusobnih odnosa riječi uz identifikaciju gramatičkih

uloga pojedinih riječi, dok se semantička analiza odnosi na tumačenje značenja riječi i njihovih kombinacija, uključujući razumijevanje konteksta i konotaciju tih riječi [1,2,3,4,5].

2.1.2. Važnost algoritama za mjerenje sličnosti teksta

Algoritmi za mjerenje sličnosti teksta predstavljaju ključni instrument u analizi i organizaciji informacija, pružajući mogućnost preciznog i brzog pretraživanja obimnih skupova podataka kako bi se identificirale relevantne informacije. Ova sposobnost ima značajnu primjenu u različitim kontekstima, pridonoseći poboljšanju učinkovitosti pretraživanja i analize informacija.

Identifikacija sličnosti među dokumentima, potpomognuta algoritmima za mjerenje sličnosti teksta, ima ključnu ulogu u klasifikaciji dokumenata. Ovaj proces postaje od iznimne važnosti u organizaciji i strukturiranju informacija, čime se olakšava brza i precizna pristupačnost relevantnih sadržaja. Kroz kvalitetnu klasifikaciju, korisnici mogu efikasnije upravljati informacijama, što se posebno ističe u radu s velikim skupovima podataka.

Dodatno, algoritmi za mjerenje sličnosti teksta imaju ključnu ulogu u suzbijanju širenja lažnih informacija. Identifikacija varijacija između autentičnih i lažnih vijesti omogućuje stvaranje sustava koji doprinosi borbi protiv dezinformacija. Ova tehnologija postaje neophodna u suvremenom informacijskom okruženju gdje je razlučivanje istinitih informacija od netočnih ključno za održavanje pouzdanosti i integriteta informacijskog prostora. Algoritmi za mjerenje sličnosti teksta pružaju sredstva za analizu i otkrivanje obrazaca, čime se podupire transparentnost informacija i jača kapacitet razlučivanja pouzdanih izvora od onih manje vjerodostojnih [5,6,7,8,9,10].

2.2. Detaljan pregled algoritama

U idućem poglavlju opisano je pet algoritama za mjerenje sličnosti teksta. Dan je detaljan prikaz algoritama koji koriste različite pristupe i tehnike za rješavanje problema određivanja sličnosti dvaju tekstova.

2.2.1 Levenshtein udaljenost i Damerau-Levenshtein udaljenost

Levenshtein udaljenost je mjera sličnosti između dva niza znakova koja uzima u obzir broj umetanja, brisanja i zamjena znakova potrebnih da se jedan niz pretvori u drugi. Ova metrika se često koristi u računarskoj lingvistici za mjerenje sličnosti između riječi i rečenica, za korekciju pravopisa i prepoznavanje govornih lica. Levenshtein udaljenost je prvi put definirao

sovjetski matematičar Vladimir Levenshtein u svom radu 1965. godine.On je razmatrao problem pretraživanja riječi u tekstu. Predložio je da se udaljenost između dvije riječi mjeri kao minimum broja operacija koje su potrebne za transformaciju jedne riječi u drugu.

Levenshteinov algoritam mjeri sličnost između dva niza znakova na temelju broja operacija koje su potrebne za transformiranje jednog niza u drugi. Ove operacije mogu biti zamjena, dodavanje ili uklanjanje znaka, a najčešće se računaju korištenjem dinamičkog programiranja [11, 12, 13].

Dinamičko programiranje je tehnika koja složene probleme raščlanjuje na manje, jednostavnije probleme. Za računanje Levenshteinove udaljenosti između dvije riječi, koristi se matrica dimenzija $(B+1)*(C+1)$, gdje su B i C duljine dvije riječi. Matrica se inicijalizira na određeni način, a cilj je popuniti cijelu matricu počevši od gornjeg lijevog kuta i na kraju donji desni kut matrice daje Levenshteinovu udaljenost. Svaka ćelija u matrici predstavlja minimalni broj operacija potreban da se pretvori prefiks jednog niza u prefiks drugog. Na primjer, ako uspoređujemo riječi "kotao" i "auto", matrica bi pokazala korake potrebne za pretvaranje "kot" u "au", "kota" u "aut", i tako dalje.

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & ako\ je \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & inače. \end{cases} \quad (1)$$

gdje je:

- $lev_{a,b}(i-1,j)+1$- udaljenost nakon brisanja znaka iz a (ili umetanja u b)
- $lev_{a,b}(i,j-1)+1$- udaljenost nakon umetanja znaka u a (ili brisanja iz b)
- $lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}$- udaljenost nakon zamjene znaka (ako su a_i i b_j različiti, inače se ne dodaje ništa) [13].

Formula (1) započinje s osnovnim slučajem gdje su jedan ili oba niza prazni (duljina 0), u kojem slučaju je udaljenost jednaka duljini drugog niza (jer su sve operacije ili umetanja ili brisanja). Zatim rekurzivno uspoređuje svaki znak jednog niza s svakim znakom drugog niza, uzimajući u obzir troškove umetanja, brisanja i zamjene znakova. Ova rekurzivna formula temelj je algoritma koji se često implementira koristeći dinamičko programiranje kako bi se poboljšala efikasnost izračuna, posebno za duže nizove.

Slijedi primjer u kojem se pomoću algoritma Levenshteinove udaljenosti izračunava sličnost između riječi PRIJATELJ i PRIJEDLOG. Izvor, odnosno riječ od koje se kreće, je PRIJATELJ. Cilj je različitim transformacijama koje uključuju umetanje, brisanje i zamjenu znakova od izvorne riječi doći do odredišta, što je u ovom slučaju riječ PRIJEDLOG.

Tablica 2.1 Levenshteinova udaljenost na primjeru riječi prijatelj i prijedlog

Izvor	P	R	I	J	A	T	E	L	J
Odredište	P	R	I	J	E	D	L	O	G
Izmjene	-	-	-	-	A->E	T->D	E->L	L->O	J->G

Formula za izračun sličnosti teksta je:

$$Sličnost\ teksta = 1 - \frac{Broj\ promjena}{Duljina\ najdulje\ riječi} \qquad (2)$$

gdje je:

- broj promjena je broj izmjena potrebnih da se jedna riječ pretvori u drugu (uključuje umetanje, brisanje ili zamjenu znakova)
- duljina najdulje riječi je izmjerena duljina najduže riječi od dvije riječi koje se uspoređuju

U Tablici 2.1 vidljivo je kako je između riječi PRIJATELJ i PRIJEDLOG potrebno je napraviti pet izmjena kako bi iz riječi PRIJATELJ dobili riječ PRIJEDLOG, a najduža riječ ima devet znakova.

$$Sličnost\ teksta = 1 - \frac{5}{9} = 1 - 0{,}56 = 0{,}44$$

Temeljem toga, riječ PRIJATELJ je samo 44% slična riječi PRIJEDLOG.

Prilikom izračuna udaljenosti popunjava se već prije spomenuta matrica pomoću dinamičkog programiranja koja za zadane riječi iz primjera izgleda kao na Tablici 2.1. Iz matrice se može vidjeti da je Levenshteinova udaljenost između PRIJATELJ i PRIJEDLOG 5, što se nalazi u donjem desnom kutu matrice. Ova matrica pokazuje minimalni broj izmjena (umetanja, brisanja ili zamjena znakova) potrebnih da se jedna riječ pretvori u drugu.

Tablica 2.2 Matrica Levenstheinove udaljenosti za riječi prijatelj i prijedlog

		P	R	I	J	A	T	E	L	J
	0	1	2	3	4	5	6	7	8	9
P	1	0	1	2	3	4	5	6	7	8
R	2	1	0	1	2	3	4	5	6	7
I	3	2	1	0	1	2	3	4	5	6
J	4	3	2	1	0	1	2	3	4	5
E	5	4	3	2	1	1	2	2	3	4
D	6	5	4	3	2	2	2	3	3	4
L	7	6	5	4	3	3	3	3	3	4
O	8	7	6	5	4	4	4	4	4	4
G	9	8	7	6	5	5	5	5	5	5

Damerau-Levenshteinova udaljenost je varijacija Levenshteinove udaljenosti, ali uz dodatnu operacijuzamjenu susjednih znakova. To znači da, uz umetanja, brisanja, i zamjene, Damerau-Levenshteinova udaljenost također uzima u obzir transpoziciju dva susjedna znaka. Ova metoda je posebno korisna u situacijama gdje su tipične greške u pisanju transpozicije, kao što je čest slučaj u pisanju na tipkovnici.

Dakle, ključna razlika između ove dvije metrike je uključivanje transpozicije kao operacije. U nekim slučajevima, Damerau-Levenshteinova udaljenost može dati precizniji uvid u sličnost dva niza, posebno u kontekstu ljudskih grešaka u pisanju [11, 12, 13].

U nastavku je pseudokod za izračun Levenshteinove udaljenosti koji se može implementirati u bilo kojem programskom jeziku.

```
funkcija LevenshteinovaUdaljenost(izvor, odredište):
    m = duljina(odredište) + 1
    n = duljina(izvor) + 1
    matrica = polje veličine m x n    za i od 0 do m - 1:
        matrica[i][0] = i
    za j od 0 do n - 1:
   matrica[0][j] = j
    za i od 1 do m - 1:
        za j od 1 do n - 1:
```

```
            ako je odredište[i - 1] jednako izvor[j - 1]:
                matrica[i][j] = matrica[i - 1][j - 1]
            inače:
                matrica[i][j] = 1 + minimum(matrica[i - 1][j],
                                    matrica[i][j - 1],
                                     matrica[i - 1][j -1]
                                         )
vrati matrica[m - 1][n - 1]
```

2.2.2. Jaro i Jaro-Winkler sličnost

Jaro sličnost je mjera sličnosti između dva niza znakova s_1 i s_2 koja se kreće od 0 do 1, gdje 1 znači da su nizovi jednaki, a 0 znači da nema sličnosti između dva niza. Jaro sličnost se računa pomoću formule koja koristi broj podudaranja znakova i broj transpozicija između dva niza. Posebno je korisna u situacijama gdje se nizovi mogu razlikovati u smislu transpozicije znakova, što je često u slučajevima kao što su tipografske greške.

$$sim_j = \begin{cases} 0 & ako\ je\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & inače. \end{cases} \quad (3)$$

gdje je:

- $|m|$ broj podudarnih znakova
- t broj transpozicija (pola broja znakova koji se podudaraju, ali su na različitim pozicijama u dva niza)
- $|s_1|$i $|s_2|$su duljine nizova znakova s_1 i s_2 [14].

Znakovi se smatraju podudarnima ako se nalaze unutar udaljenosti koja je manja od

$$\lfloor \frac{\max(|s1|,|s2|)}{2} \rfloor - 1.$$

Formula (3) uzima u obzir broj zajedničkih znakova, duljinu nizova i broj transpozicija kako bi izračunala sličnost između niza A i niza B. Rezultat će biti vrijednost između 0 (potpuno različiti nizovi) i 1 (potpuno isti nizovi). Jaro mjera sličnosti posebno se fokusira na broj podudarnih znakova i broj transpozicija između dva niza.

Dvije ključne komponente prilikom izračuna Jaro sličnosti su podudaranje znakova i transpozicija. Dva znaka iz nizova s_1 i s_2 su podudarna ako su isti i nisu udaljeniji od polovice duljine dužeg niza minus jedan. Na taj je način omogućena određena fleksibilnost pri usporedbi nizova koji mogu imati sitne razlike u rasporedu znakova. Nakon identifikacije podudarnih znakova broji se broj transpozicija koje se javljaju kada su dva podudarna znaka u različitom redoslijedu u dva niza. Broj transpozicija se računa kao polovina broja takvih neslaganja.

Jaro-Winkler sličnost je proširenje Jaro mjere sličnosti, koje daje veću težinu podudaranju na početku niza. Ova metrika pretpostavlja da je veća vjerojatnost da će se greške dogoditi prema kraju nizova. Jaro-Winkler sličnost koristi faktor prefiksa koji daje točniji odgovor kada nizovi imaju zajednički prefiks do određene maksimalne duljine.

$$sim_w = sim_j + lp(1 - sim_j) \qquad (4)$$

gdje je:

- sim_j Jaro sličnost između s_1 i s_2
- l je duljina zajedničkog prefiksa na početku nizova do maksimalno četiri znaka
- p je konstanta skaliranja (uobičajeno se koristi vrijednost 0.1) [14].

Ova se mjera sličnosti temelji na pretpostavci da su greške u tipkanju ili transkripciji češće prema kraju riječi ili izraza.

Jaro-Winkler sličnost uključuje prefiksnu skalarnu komponentu koja dodaje bonus bodove ako nizovi imaju zajednički prefiks do maksimalne duljine od četiri znaka. Za konstantu skaliranja p obično se koristi vrijednost od 0.1, ali se može prilagoditi ovisno o potrebama specifične primjene. Veća vrijednost p daje veću važnost prefiksu.

U nastavku je dan isti primjer za izračun sličnosti između riječi PRIJATELJ i PRIJEDLOG, no ovoga puta sličnost se računa pomoću algoritma Jaro i Jaro-Winkler sličnosti.

Koraci za izračun Jaro i Jaro-Winkler sličnosti dani su u nastavku teksta.

1. Odrediti zajedničke znakove. zajednički znakovi su oni koji se pojavljuju u obje riječi unutar određenog raspona.

Tablica 2.3 . Izračun broja zajedničkih znakova kod Jaro-Winkler sličnosti

Pozicija	Riječ 1 („PRIJATELJ“)	Riječ 2 („PRIJEDLOG)
1	P	P
2	R	R
3	I	I
4	J	J

Prema Tablici 2.3 vidljivo je kako su zajednički znakoviP, R, I, J (četiri znaka).

2. Izračunati broj nesukladnih znakova na istom indeksu i usporediti pozicije znakova iz prethodnog koraka.

Tablica 2.4 Broj nesukladnih znakova kod izračuna Jaro-Winkler sličnosti

Znakovi	Nesukladni znakovi
A & E	Da
T & D	Da
E & L	Da
L & O	Da
J & G	Da

Kako je prikazano u Tablici 2.4, broj nesukladnih znakova je pet.

3. Izračunati broj transpozicija, a to je polovica broja nesukladnih znakova.

$$Transpozicije\ (t) = \frac{5}{2} = 3$$

4. Izračunati Jaro, Jaro-Winkler sličnost prema formuli (3).

$$sim_j = \frac{1}{3}\left(\frac{4}{9} + \frac{4}{9} + \frac{4-3}{4}\right) = 0{,}3796$$

$$sim_w = sim_j + l * p * \left(1 - sim_j\right) = 0{,}3796 + 4 * 0{,}1 * (1 - 0{,}3796) = 0{,}6278$$

Vidljivo je kako u riječi PRIJATELJ i PRIJEDLOG postoje četiri ista znaka i potrebno je napraviti tri transpozicije. Temeljem toga, sličnost između riječi je samo 37,96% kod Jaro sličnosti, a 62,78% prilikom Jaro-Wrinkler sličnosti [15,16,17].

U nastavku je pseudokod za izračun Jaro sličosti između dva niza.

```
    funkcija JaroSličnost(izvor, odredište):
        duljina_izvor = duljina(izvor)
        duljina_odredište = duljina(odredište)

        ako je duljina_izvor == 0 i duljina_odredište == 0:
            vrati 1.0
        ako je duljina_izvor == 0 ili duljina_odredište == 0:
            vrati 0.0

        maks_distanca = (maks(duljina_izvor, duljina_odredište) //
2) - 1

        zajednički_znakovi = 0
        transpozicije = 0

        oznaka_izvor = niz duljine duljina_izvor inicijaliziran s
False
        oznaka_odredište = niz duljine duljina_odredište
inicijaliziran s False

        za i od 0 do duljina_izvor - 1:
            početak = maks(0, i - maks_distanca)
            kraj = min(duljina_odredište, i + maks_distanca + 1)

            za j od početak do kraj - 1:
                ako je odredište[j] == izvor[i] i
oznaka_odredište[j] == False:
                    oznaka_izvor[i] = True
                    oznaka_odredište[j] = True
                    zajednički_znakovi += 1
                    prekini

        ako zajednički_znakovi == 0:
```

```
            vrati 0.0

        k = 0
        za i od 0 do duljina_izvor - 1:
            ako oznaka_izvor[i]:
                dok oznaka_odredište[k] == False:
                    k += 1
                ako izvor[i] != odredište[k]:
                    transpozicije += 1
                k += 1

        transpozicije //= 2

        sim_j = (zajednički_znakovi / duljina_izvor +
                 zajednički_znakovi / duljina_odredište +
                 (zajednički_znakovi - transpozicije) /
zajednički_znakovi) / 3.0

        vrati sim_j
```

2.2.3 Smith-Waterman sličnost

Smith-Waterman sličnost je mjera sličnosti između dvije sekvence. Ova mjera se koristi za pronalaženje lokalnih poravnanja između sekvenci, što znači da se uspoređuju samo dijelovi sekvenci koji su slični. Ova mjera sličnosti koristi dinamičko programiranje za izračunavanje poravnanja. Algoritam je nazvan po svojim tvorcima Temple F. Smith i Michael S. Waterman, koji su ga prvi put opisali 1981. godine. Ovaj algoritam je varijacija na globalno poravnanje koje pruža Needleman-Wunsch algoritam. Smith-Waterman se fokusira na identificiranje sličnih regija unutar većih sekvenci koje možda nisu u potpunosti slične.

Formula za popunjavanje matrice poravnanja u Smith-Waterman algoritmu je sljedeća:

$$H(i,j) = max\begin{cases} 0 \\ H(i-1,j-1) + bod\ za\ podudaranje\ ili\ kazna\ za\ neslaganje \\ H(i-1,j) + kazna\ za\ prazninu \\ H(i,j-1) + kazna\ za\ prazninu \end{cases} \quad (5)$$

gdje $H(i,j)$ predstavlja bod u ćeliji matrice poravnanja za i-ti znak u prvoj sekvenci i j-ti znak u drugoj sekvenci [18].

Algoritam radi na sljedeći način:

1. Inicijalizira se matrica veličine *(m+1) x (n+1)*, gdje su *m* i *n* duljine sekvenci koje se uspoređuju.
2. Svaki element matrice se inicijalizira na nula kako je prikazano u Tablici 2.5.

Tablica 2.5 Inicijalizacija matrice kod Smith-Waterman sličnosti

		P	R	I	J	A	T	E	L	J
	0	0	0	0	0	0	0	0	0	0
P	0									
R	0									
I	0									
J	0									
E	0									
D	0									
L	0									
O	0									
G	0									

3. Svaki element matrice se izračunava pomoću sljedeće formule:

```
score(i, j) = max(
    0,score(i-1, j-1) + match(i, j),
  score(i-1, j) - gap_penalty,
score(i, j-1) - gap_penalty
)
```

gdje $score(i,j)$ predstavlja vrijednost elementa matrice na poziciji (i,j), match (i,j)predstavlja vrijednost podudaranja između znakova na pozicijama *i* i *j*, a *gap_penalty* predstavlja kaznu za otvaranje novog proreza. Popunjava se matrica koristeći nagradu za podudaranje (+2), kaznu za neslaganje (-1) i kaznu za umetanje/brisanje (-1). Matrica je prikazana u idućoj Tablici 2.6.

Tablica 2.6 Matrica izračuna Smith-Waterman sličnosti na primjeru riječi prijatelj i prijedlog

		P	R	I	J	A	T	E	L	J
	0	0	0	0	0	0	0	0	0	0
P	0	2	1	0	0	0	0	0	0	0
R	0	1	4	3	2	1	0	0	0	0
I	0	0	3	6	5	4	3	2	1	0
J	0	0	2	5	8	7	6	5	4	3
E	0	0	1	4	7	6	5	4	3	6
D	0	0	0	3	6	5	4	3	5	4
L	0	0	0	2	5	4	3	2	4	3
O	0	0	0	1	4	3	2	1	3	2
G	0	0	0	0	3	2	1	0	2	1

Pronalazi se element matrice s najvećom vrijednošću. To je krajnji rezultat poravnanja.

4. Koristeći matricu, može se rekonstruirati poravnanje [19].

Pogledom na Tablicu 2.6, lako se uoči kako je najveća vrijednost u matrici osam, a za pretvorbu Smith-Waterman sličnosti u postotak koristi se sljedeća formulu:

$$Postotna\ sličnost = \left(\frac{Smith{-}Waterman\ sličnost}{maksimalna\ moguća\ sličnost}\right) \times 100 \qquad (6)$$

Dobivena Smith-Waterman sličnost je osam, a maksimalna moguća sličnost je $9x2 = 18$, temeljem toga, sličnost između riječi je 44,44%.

Četiri su ključne karakteristike Smith-Waterman algoritma: lokalno poravnanje, sustav bodovanja, matrica poravnanja i „traceback“. Smith-Waterman lokalnim poravnanjem, za razliku od globalnog poravnanja koje pokušava poravnati cijele sekvence, identificira lokalne regije s visokom razinom sličnosti. To omogućava otkrivanje funkcionalno ili strukturno važnih podsekvenci unutar većih sekvenci.

Algoritam koristi sustav bodovanja koji dodjeljuje bodove za podudaranje, kaznu za neslaganje (mismatch) i kaznu za prazninu (gap). Sustav bodovanja može se prilagoditi

specifičnim potrebama istraživanja. Također, Smith-Waterman koristi matricu poravnanja za izračun najboljeg lokalnog poravnanja. Svaka ćelija u matrici predstavlja maksimalni broj bodova koji se može postići poravnanjem sekvenci do te točke. Nakon izgradnje matrice, algoritam za izračun Smith-Waterman sličnosti izvodi proces nazvan "traceback" počevši od ćelije s najvišim bodom. „Traceback“ prati put natrag kroz matricu koji je rezultirao najvišim bodom, rekonstruirajući najbolje lokalno poravnanje [20,21].

U nastavku je pseudokod za izračun Smith-Waterman sličosti između dvije sekvence.

```
funkcija SmithWaterman(izvor, odredište):
    nagrada = 2
    kazna = -1

    m = duljina(izvor)
    n = duljina(odredište)

    matrica = polje veličine (m + 1) x (n + 1) inicijalizirano
    s 0

    max_vrijednost = 0

    za i od 1 do m:
        za j od 1 do n:
            ako izvor[i - 1] == odredište[j - 1]:
                dijagonalno = matrica[i - 1][j - 1] + nagrada
            inače:
                dijagonalno = matrica[i - 1][j - 1] + kazna

            lijevo = matrica[i][j - 1] + kazna
            gore = matrica[i - 1][j] + kazna

            matrica[i][j]    =    maksimum(0,    dijagonalno,
lijevo,gore)

            ako matrica[i][j] > max_vrijednost:
                max_vrijednost = matrica[i][j]

    vrati max_vrijednost
```

2.2.4 Kosinusna sličnost

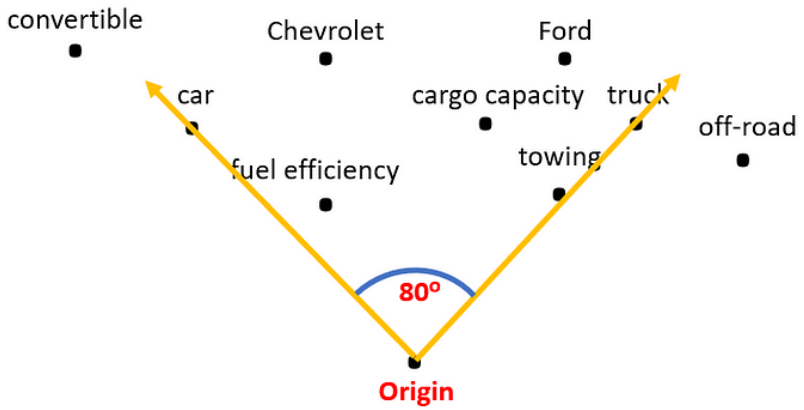
Kosinusna sličnost je mjera sličnosti između dva ne-nula vektora u višedimenzionalnom prostoru koja se koristi u raznim područjima, kao što su sustavi preporuka, pretraživanje informacija, obrada prirodnog jezika, računalni vid i strojno učenje, za mjerenje sličnosti između dokumenata, tekstova ili različitih setova podataka. Kosinusna sličnost mjeri kosinus kuta između dvaju vektora i daje vrijednost između minus jedan i jedan. Vrijednost jedan znači da su vektori jednaki, vrijednost nula znači da su vektori potpuno različiti, a vrijednost minus jedan znači da su vektori suprotni.

Matematički, za dva vektora A i B, kosinusna sličnost cos(θ) je definirana kao:

$$kosinusna\ sličnost = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} \qquad (7)$$

gdje:

- $A \cdot B$ predstavlja skalarni produkt vektora A i B
- $\|A\|\ i\ \|B\|\|$ su norme (duljine) vektora A i B respektivno [22].



Slika 2.1. Primjer kosinusne sličnosti,
Izvor: https://medium.com/@arjunprakash027/understanding-cosine-similarity-a-key-concept-in-data-science-72a0fcc57599

Na Slici 2.1. su prikazana dva vektora koji izlaze iz iste točke, označene kao "Origin". Kako bi se izračunala kosinusna sličnost između ova dva vektora, prvo se identificira kut između njih (u ovom slučaju označen kao 80°), a zatim se računa kosinus tog kuta. Na slici su navedeni pojmovi koji bi mogli predstavljati karakteristike automobila i kamiona, npr. "convertible" (kabriolet), "fuel efficiency" (učinkovitost goriva), "cargo capacity" (kapacitet tereta),

"towing" (vuča) i "off-road" sposobnosti. Ako gledamo automobile kao vektore u prostoru karakteristika, ovi pojmovi mogu predstavljati osi na tom prostoru ili vrijednosti na određenim osima za svaki vektor. Na primjer, automobil bi mogao imati visoku vrijednost na osi "fuel efficiency", dok bi kamion mogao imati visoku vrijednost na osi "cargo capacity" [22].

Kosinusna sličnost bi u ovom kontekstu mogla pomoći da se kvantificira koliko su dva vozila slična u pogledu svojih karakteristika, što može biti korisno u raznim primjenama poput segmentacije tržišta, ciljanog oglašavanja ili preporuka proizvoda [6, 22].

U nastavku je pseudokod za izračun kosinusne sličnosti između dvije sekvence.

```
    funkcija kosinusnaSličnost(vektorA, vektorB):
        skalarniProdukt = 0
        magnitudaA = 0
        magnitudaB = 0

        n = duljina(vektorA)

        za i od 0 do n - 1:
            skalarniProdukt = skalarniProdukt + (vektorA[i] *
vektorB[i])
            magnitudaA = magnitudaA + (vektorA[i] * vektorA[i])
            magnitudaB = magnitudaB + (vektorB[i] * vektorB[i])

        magnitudaA = korijen(magnitudaA)
        magnitudaB = korijen(magnitudaB)

        ako magnitudaA == 0 ili magnitudaB == 0:
            vrati 0

        kosinusnaSličnost = skalarniProdukt / (magnitudaA *
magnitudaB)
        vrati kosinusnaSličnost
```

2.2.5 N-gram sličnost

N-gram sličnost je metoda sličnosti koja se koristi za analizu i usporedbu tekstualnih podataka u obradi prirodnog jezika (engl. *natural language processing* – NLP). N-grami su

sekvence od N uzastopnih elemenata (tipično riječi ili znakova) izvučene iz teksta. Metoda N-gram sličnost mjeri sličnost između dvaju tekstova temeljem preklapanja njihovih N-grama. Na primjer, 2-gram (ili bigram) za string "hello" su "he", "el", "ll" i "lo".

Sličnost se može izračunati pomoću formule:

$$\frac{2 \cdot |\, X \cap Y \,|}{|\, X \,| + |\, Y \,|} \qquad (8)$$

gdje X i Y predstavljaju skupove n-grama za dva niza koje uspoređujemo. Formula daje vrijednost između nula i jedan, gdje jedan znači da su nizovi jednaki, a nula znači da nemaju zajedničkih n-grama.

Primjerice, ako uspoređujemo nizove "kitten" i "sitting", tada se 2-grami za svaki niz mogu izračunati na sljedeći način:

- "kitten"- "ki", "it", "tt", "te", "en"
- "sitting"- "si", "it", "tt", "ti", "in", "ng"

Zajednički 2-grami su "it" i "tt", pa je veličina presjeka $|X \cap Y| = 2$. Veličina unije $|X| + |Y| = 9$. Stoga je N-gram sličnost između "kitten" i "sitting" koristeći formulu (8) jednaka:

- $\frac{2 \cdot |X \cap Y|}{|X| + |Y|} = 2 \cdot \frac{2}{9} \approx 0.4444$

Dobivena vrijednost ukazuje na to da su nizovi "kitten" i "sitting" samo djelomično slični [23, 24, 25].

U nastavku je pseudokod za izračun N-gram sličnosti između dva niza.

```
funkcija generirajNGrame(riječ, N):
    nGramovi = prazna lista

    za i od 0 do duljina(riječ) - N:
        nGram = riječ[i : i + N]
        dodaj nGram u nGramovi

    vrati nGramovi

funkcija nGramSličnost(riječ1, riječ2, N):
```

```
        nGramovi1 = generirajNGrame(riječ1, N)
        nGramovi2 = generirajNGrame(riječ2, N)

        zajedničkiNGramovi = 0

        za svaki nGram u nGramovi1:
            ako nGram je u nGramovi2:
                zajedničkiNGramovi = zajedničkiNGramovi + 1
                ukloni nGram iz nGramovi2

        ukupniNGramovi = (duljina(nGramovi1) + duljina(nGramovi2) +
zajedničkiNGramovi) / 2

        ako ukupniNGramovi == 0:
            vrati 0

        sličnost = zajedničkiNGramovi / ukupniNGramovi
        vrati sličnost
```

3. Analiza i primjena algoritama

Svaki od algoritama za mjerenje sličnosti teksta ima svoje jedinstvene karakteristike, prednosti i ograničenja koje su analizirane u ovom poglavlju.

3.1. Levenshteinova i Damerau-Levenshteinova udaljenost

Levenshteinova i Damerau-Levenshteinova udaljenost su pokazale visoku efikasnost u situacijama gdje su male varijacije u tekstu kao što su tipografske greške ili gramatičke varijacije. Međutim, njihova učinkovitost smanjuje se s povećanjem dužine teksta [13].

Najčešće područje primjene ovih algoritama uključuje otkrivanje i automatsko ispravljanje tipografskih i pravopisnih grešaka u tekstu. Također, koriste se za identificiranje i uklanjanje dupliciranih ili sličnih podataka unutar velikih baza podataka. Osim toga, analiziraju sličnosti i razlike između različitih verzija dokumenata. Primjenjuju se i kod ispravljanja pogrešaka u transkripciji govora na temelju usporedbe standardnim oblicima riječi i poboljšanjem razumijevanja varijacija u izgovoru, usklađivanja prijevoda s izvornim tekstom radi osiguravanja točnosti, usporedbe genetskih i proteinskih sekvenci te identifikaciju plagijata.

Prostorna složenost ovih algoritama je $O(m \times n)$ jer koristimo matricu veličine $m \times n$ za pohranu međurezultata, gdje su m i n duljine odredišne i izvorne riječi plus jedan.

Vremenska složenost algoritama je $O(m \times n)$ jer svaki element u matrici zahtijeva konstantno vrijeme za izračunavanje (usporedba i tri pristupa memoriji) te postoji ukupno $m \times n$ elemenata koje trebamo popuniti.

Konačno, prilikom korištenja algoritma za izračunavanje Levenshteinove udaljenosti formira se matrica za pohranu rezultata podproblema. Algoritam ima vremensku i prostornu složenost od $O(m \times n)$. To znači da vrijeme i prostor potrebni za izvođenje algoritma rastu linearno s umnoškom duljina dviju riječi koje se uspoređuju.

3.2. Jaro i Jaro- Winkler sličnost

Jaro i Jaro-Winkler sličnost su se istaknuli u scenarijima gdje je važna sličnost početnih segmenata, poput usporedbe imena ili naslova. Ovi algoritmi su bili manje učinkoviti u dužim tekstovima gdje su varijacije raspršene [27].

Primjena algoritma za izračun Jaro i Jaro-Winkler sličnosti obuhvaća identificiranje i uklanjanje dupliciranih zapisa u bazama podataka kao što su CRM sustavi, gdje različiti unos podataka može rezultirati dupliciranim zapisima iste osobe. Također, koristi se za uspoređivanje i povezivanje srodnih podataka iz različitih izvora, primjerice u procesima integracije podataka. Primjena ovih algoritama korisna je i kod identificiranja i ispravljanja pogrešaka u tekstu koji je generiran prepoznavanjem govora ili prilikom pravopisnih provjera. Također, ovi se algoritmi koriste za uspoređivanje nizova kako bi se pronašle sličnosti u imenima proizvoda, naslovima knjiga, filmova i slično. Pomažu i u razumijevanju i povezivanju srodnih tema ili diskusija, čak i kada postoje varijacije u imenovanju ili tipografske greške na društvenim mrežama. Na kraju, omogućavaju povezivanje sličnih proizvoda ili sadržaja usprkos malim varijacijama u nazivima.

Prostorna složenost Jaro algoritma je $O(m \times n)$ zbog korištenja dvaju pomoćnih nizova, dok je vremenska složenost Jaro algoritma $O(m \times n)$ zbog dvostruke petlje koja prolazi kroz sve moguće parove znakova unutar dozvoljenog raspona.

Algoritam ima vremensku i prostornu složenost $O(m \times n)$, što ga čini efikasnim za usporedbu kratkih do srednje dugih riječi.

3.3. Smith- Waterman sličnost

Smith–Waterman sličnost pokazala je izvanredne rezultate u identificiranju sličnih segmenata unutar dužih tekstova, ali s većom potrošnjom vremena i resursa. Ključni je alat u bioinformatici zbog svoje sposobnosti da pruža detaljno i precizno lokalno poravnanje, omogućujući istraživačima da dublje razumiju složene biološke sekvence. Algoritam je zahtjevan u pogledu računalnih resursa, posebno za duže sekvence, ali njegova preciznost i detaljnost čine ga nezamjenjivim alatom u molekularnoj biologiji i srodnim disciplinama [28].

Algoritam za izračun Smith-Waterman sličnosti primjenjuje se kod analize genoma i proteoma te kod identificiranja sličnih regija unutar DNA, RNA ili proteinskih sekvenci što može pomoći u otkrivanju genetskih veza i funkcionalnih sličnosti. Također, koristi se u istraživanju evolucijskih odnosa, pomažući u razumijevanju evolucijskih odnosa između različitih vrsta ili unutar populacija. Ovaj algoritam može pomoći i u identifikaciji bolesti, omogućujući prepoznavanje mutacija ili promjena u sekvencama koje su povezane s određenim bolestima. Dodatno, koristi se za analizu teksta, pronalazeći slične ili identične sekvence unutar

većih tekstova. Pomaže i u forenzičkoj analizi, identificirajući i uspoređujući dijelove teksta u pravnim istraživanjima. Na kraju, ovaj algoritam je koristan u akademskim istraživanjima, posebno pri detekciji plagijata ili dupliciranja u znanstvenim radovima.

Prostorna složenost algoritma je $O(m \times n)$ zbog korištenja matrice dimenzija $(m + 1) \times (n + 1)$, gdje su m i n duljine izvorne i odredišne riječi.

Vremenska složenost algoritma je $O(m \times n)$ jer svaki element matrice zahtijeva konstantno vrijeme za izračunavanje, a postoji ukupno $(m + 1) \times (n + 1)$ elemenata matrice koje je potrebno popuniti.

3.4. Kosinusna sličnost

Kosinusna sličnost se istaknula u analizi velikih skupova podataka, posebno u aplikacijama poput pretraživanja informacija i analize sentimenta gdje je potrebno analizirati sličnost na razini cijelog dokumenta. Izuzetno je korisna u mnogim aspektima obrade prirodnog jezika i analize podataka, pružajući intuitivan i efikasan način za mjerenje sličnosti između objekata. Njena sposobnost da izolira sličnost u smislu orijentacije, a ne veličine, čini je idealnom za primjene gdje je bitno usporediti obrasce učestalosti ili prisutnosti značajki [29].

Primjena algoritma za izračun kosinusne sličnosti obuhvaća sustave preporuka gdje se preporučuju slični proizvodi, filmovi, knjige ili članci na temelju preferencija korisnika. Također se koristi za pretraživanje dokumenata, pronalazeći i preporučujući slične dokumente ili članke unutar velike baze podataka. Ovaj algoritam je koristan u grupiranju i klasifikaciji teksta. Kosinusna sličnost može se koristiti i za analizu sentimenta, uspoređujući emocionalni ton i sadržaj u različitim izvorima. Također se primjenjuje u analizi mrežnih vijesti, gdje se prati i uspoređuje način izvještavanja o istim događajima u različitim medijima.

Prostorna složenost algoritma je $O(1)$ jer koristimo konstantnu količinu dodatne memorije za pohranu nekoliko varijabli bez obzira na veličinu ulaznih vektora. Vremenska složenost algoritma je $O(n)$ jer svaki element vektora zahtijeva konstantno vrijeme za izračunavanje, a postoji ukupno (n) elemenata koje trebamo obraditi.

3.5. N-gram sličnost

N-gram sličnost je koristan alat u analizi fonetskih i ortografskih sličnosti na mikro razini, što je posebno korisno u jezičnim analizama i leksikografskim studijama. U marketingu i praćenju društvenih medija koristi se za analizu sentimenta korisničkih recenzija i komentara na društvenim mrežama. Ova analiza omogućuje kompanijama da dobiju uvid u percepciju svojih proizvoda i usluga te da brzo reagiraju na povratne informacije korisnika. N-gram sličnost je moćan alat u NLP-u za analizu i usporedbu tekstova. Njegova primjena seže od lingvistike do analize podataka, omogućujući duboko razumijevanje i usporedbu tekstualnih uzoraka. Izbor duljine N-grama ovisi o specifičnim potrebama i karakteristikama analize [30].

Algoritam za izračun N-gram sličnosti primjenjuje se u raznim područjima. Jedna od ključnih primjena je detekcija plagijata, gdje se dokument uspoređuje s drugima kako bi se pronašla neautorizirana sličnost. U prevoditeljskim uslugama koristi se za usporedbu i analizu prevedenih tekstova s izvornicima. Također se primjenjuje u sustavima preporuka, gdje pomaže preporučiti sličan sadržaj temeljen na tekstualnoj analizi. U lingvističkoj analizi koristi se za istraživanje i usporedbu stilova pisanja između različitih autora ili vremenskih razdoblja. Algoritam se koristi i u automatiziranom prepoznavanju govora, gdje poboljšava točnost prepoznavanja analizom konteksta i učestalosti sekvenci. Sigurnosne provjere također koriste ovaj algoritam za analizu komunikacija s ciljem otkrivanja sumnjivog ili neautoriziranog sadržaja. U edukaciji i učenju jezika, N-gram sličnost pomaže u alatima za podršku učenju kroz analizu učestalosti i upotrebe riječi ili fraza.

Prostorna složenost je $O(m \times n)$ jer trebamo pohraniti sve N-grame za obje riječi, dok je vremenska složenost ovog algoritma je $O(m \times n)$ gdje su (n) i (m) duljine dviju riječi. Vremenska složenost je takva zbog generiranja N-grama za obje riječi i njihove usporedbe.

Istraživanje efikasnosti ovih algoritama u praksi pokazuje da je izbor algoritma ključan za specifičnu primjenu. Na primjer, u detekciji plagijata, algoritmi koji se fokusiraju na detaljnu analizu teksta, poput Levenshteinove udaljenosti, pokazali su visoku učinkovitost. S druge strane, u analizi sentimenta, algoritmi koji analiziraju globalnu strukturu teksta, kao što je Kosinusna sličnost, bili su učinkovitiji u prepoznavanju općeg tona i sentimenta. Ovi nalazi ukazuju na potrebu za ciljanom primjenom algoritama, uzimajući u obzir specifičnosti svakog pojedinog zadatka unutar računarske lingvistike. Odabir pravog algoritma ovisi o specifičnim potrebama i karakteristikama problema koji se rješavaju.

4. Web aplikacija za računanje sličnosti

Razvijanje učinkovitog alata za usporedbu tekstova može značajno unaprijediti procese usporedbe, omogućujući brže i preciznije analize. U tu svrhu, razvijena je web aplikacija koja omogućava korisnicima jednostavno uspoređivanje dva tekstualna dokumenta korištenjem različitih algoritama za mjerenje sličnosti.

Ova aplikacija pruža korisnicima intuitivno korisničko sučelje za unos tekstova ili učitavanje PDF datoteka, odabir algoritma za usporedbu i prikaz rezultata. Glavne funkcionalnosti aplikacije uključuju izračunavanje sličnosti između dva teksta korištenjem različitih algoritama, mjerenje performansi svih algoritama za usporedbu njihovih performansi te vizualni prikaz rezultata u obliku tablica.

4.1 Tehnologije korištene u razvoju

Razvoj web aplikacije za računanje sličnosti teksta zahtijevao je upotrebu niza različitih tehnologija. Svaka od ovih tehnologija odabrana je zbog svojih specifičnih prednosti i mogućnosti koje su omogućile efikasnu i pouzdanu implementaciju svih funkcionalnosti aplikacije. U nastavku su detaljno opisane tehnologije korištene u razvoju aplikacije.

Programski jezik Python je odabran za implementaciju algoritama za mjerenje sličnosti zbog svoje jednostavnosti, čitljivosti i bogate kolekcije biblioteka koje podržavaju obradu teksta i numeričke izračune. Osim toga, Python omogućuje brzu i efikasnu razvojnu okolinu, što je ključno za prototipiranje i testiranje algoritama. Glavne biblioteke korištene za implementaciju algoritama uključuju:

- ***NumPy*** - koristi se za efikasne numeričke izračune.
- ***Flask*** - mikro web framework
- ***Psutil*** - za mjerenje resursne potrošnje
- ***PyMuPDF*** - za rad s PDF datotekama

Flask je mikro web framework za Python, korišten za razvoj poslužiteljskog dijela aplikacije. Flask je odabran zbog svoje jednostavnosti, fleksibilnosti i minimalne složenosti, što ga čini idealnim za izradu malih do srednje velikih web aplikacija. Flask omogućuje brzu implementaciju server-side logike, podržava jednostavno rukovanje HTTP zahtjevima i pruža integraciju s različitim bazama podataka i drugim servisima.

Iduće korištene tehnologije su JavaScript i jQuery. Korišteni su za dinamičko upravljanje korisničkim sučeljem i komunikaciju s poslužiteljskim dijelom putem AJAX zahtjeva. JavaScript je univerzalni jezik za web razvoj koji omogućuje interaktivnost i dinamično ažuriranje sadržaja na web stranicama. jQuery, kao popularna JavaScript biblioteka, pojednostavljuje manipulaciju DOM-om, rukovanje događajima i AJAX zahtjevima, čime se ubrzava razvoj korisničkog sučelja i poboljšava korisničko iskustvo.

Za izradu korisničkog sučelja korišteni su HTML i CSS. HTML služi za strukturiranje sadržaja na web stranici, dok CSS omogućuje stiliziranje i vizualno oblikovanje tog sadržaja. Korištenje HTML-a i CSS-a omogućuje izradu preglednog i intuitivnog korisničkog sučelja.

Jedan od CSS frameworka je i Bootstrap koji je korišten za responzivni dizajn korisničkog sučelja. Odabran je zbog svoje jednostavnosti, bogatstva unaprijed definiranih komponenti i mogućnosti brzog razvoja responzivnih i mobilno-prilagođenih web stranica. Bootstrap omogućuje dosljedan izgled i osjećaj aplikacije na različitim uređajima i ekranima, čime se poboljšava korisničko iskustvo.

S obzirom da aplikacija korisnicima omogućava učitavanje PDF datoteka, potreban je i alat koji omogućava rad s njima. Jedna od biblioteka programa Python je PyMuPDF.PyMuPDF biblioteka je korištena za ekstrakciju teksta iz PDF datoteka koje korisnici mogu učitati u aplikaciju. PyMuPDF pruža brzu i efikasnu obradu PDF-ova, omogućujući preciznu ekstrakciju tekstualnog sadržaja koji se zatim koristi za izračunavanje sličnosti.

Još jedna od korisnih Pythonovih biblioteka je i Psutil. koja je korištena za mjerenje resursne potrošnje aplikacije, uključujući memorijsko opterećenje i procesorsko vrijeme. Ova biblioteka omogućuje praćenje performansi aplikacije u stvarnom vremenu, što je ključno za mjerenje performansi funkcionalnosti koja uspoređuje različite algoritme za mjerenje sličnosti.

4.2 Arhitektura aplikacije

Arhitektura web aplikacije za računanje sličnosti tekstova dizajnirana je kako bi omogućila modularnost, skalabilnost i jednostavnost održavanja. Aplikacija se sastoji od tri glavna dijela: korisničkog sučelja (engl. *frontend*), poslužiteljskog dijela (engl. *backend*) i algoritama za sličnost. U nastavku je detaljan opis svakog od ovih dijelova.

4.2.1 Korisničko sučelje (engl. *frontend*)

Korisničko sučelje predstavlja prvi kontakt korisnika s aplikacijom. Dizajnirano je tako da bude intuitivno i jednostavno za korištenje, omogućavajući korisnicima da lako pronađu potrebne funkcionalnosti. Glavne funkcionalnosti korisničkog sučelja uključuju:

- unos tekstova-korisnici mogu unijeti dva teksta u tekstualna polja
- učitavanje PDF datoteka- korisnici mogu učitati dvije PDF datoteke, iz kojih će se automatski ekstraktirati tekst
- odabir algoritma za usporedbu- korisnici mogu odabrati jedan od dostupnih algoritama za mjerenje sličnosti ili sve odjednom
- prikaz rezultata- rezultati usporedbe prikazuju se pojedinačno ili za sve algoritme u tabličnom formatu, omogućavajući korisnicima jednostavnu analizu sličnosti

Korisničko sučelje je razvijeno korištenjem HTML-a za strukturu, CSS-a za stilizaciju te JavaScript-a i jQuery-a za dinamičko upravljanje sadržajem i komunikaciju s poslužiteljskim sustavom putem AJAX zahtjeva. Bootstrap je korišten za osiguranje responzivnosti sučelja.

4.2.2 Poslužiteljski dio (engl. *backend*)

Poslužiteljski dioaplikacije implementiran je korištenjem Flask web frameworka za Python. Flask omogućava jednostavno upravljanje HTTP zahtjevima, rukovanje različitim rutama i integraciju s drugim servisima. Glavne funkcionalnosti poslužiteljskog dijela uključuju:

- obrada zahtjeva- Flask server prima zahtjeve s korisničkog sučelja, validira ulazne podatke i pokreće odgovarajuće algoritme za mjerenje sličnosti
- pokretanje algoritama- Poslužiteljski dio koristi implementirane algoritme za izračunavanje sličnosti između dva teksta ili dva ekstraktirana tekstualna sadržaja iz PDF datoteka.
- vraćanje rezultata- nakon obrade podataka, rezultati se vraćaju korisničkom sučelju u obliku JSON odgovora.

Poslužiteljski dio također koristi PyMuPDF za ekstrakciju teksta iz PDF datoteka i Psutil za mjerenje resursne potrošnje (memorija i procesorsko vrijeme).

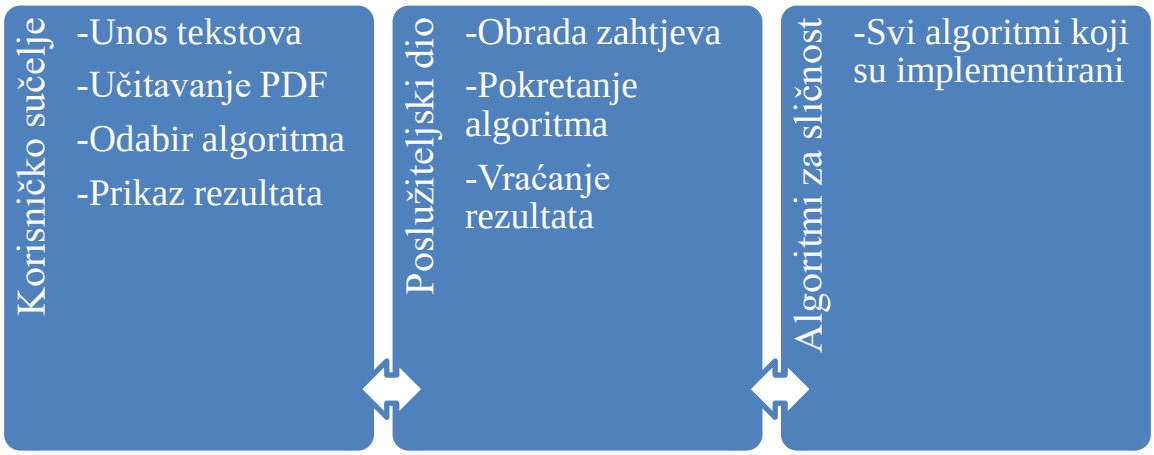
4.2.3 Algoritmi za sličnost

Algoritmi za mjerenje sličnosti implementirani su u Pythonu. Svaki algoritam ima svoju specifičnu metodu za izračunavanje sličnosti između dva teksta. Algoritmi uključuju:

- Levenshtein udaljenost- mjeri broj operacija (umetanja, brisanja, zamjene) potrebnih za pretvaranje jednog niza u drugi
- Damerau-Levenshtein udaljenost- proširenje Levenshtein udaljenosti koje uključuje transpozicije (zamjenu susjednih znakova)
- Jaro sličnost- fokusira se na podudarnost znakova i broj transpozicija između dva niza
- Jaro-Winkler sličnost- proširenje Jaro sličnosti koje daje veću težinu podudaranju početnih segmenata niza
- Smith-Waterman sličnost- algoritam za lokalno poravnanje sekvenci, koristi se za identifikaciju sličnih segmenata unutar dužih tekstova
- Kosinusna sličnost- mjeri sličnost između dva ne-nula vektora u višedimenzionalnom prostoru temeljem kosinusa kuta između njih
- N-gram sličnost- analizira tekstualne podatke na temelju preklapanja n-grama (sekvenci od n uzastopnih elemenata)

Svaki algoritam je implementiran kao zasebna funkcija unutar Python modula, omogućavajući jednostavnu integraciju i održavanje.

4.2.4 Dijagram arhitekture



Slika 4.1. Dijagram arhitekture

Arhitektura aplikacije pažljivo je dizajnirana kako bi osigurala modularnost, skalabilnost i jednostavnost korištenja. Korisničko sučelje omogućava intuitivno upravljanje aplikacijom, dok poslužiteljski dio obrađuje zahtjeve i pokreće algoritme za sličnost. Implementirani algoritmi pružaju različite metode za izračunavanje sličnosti između tekstova, čime se korisnicima omogućuje fleksibilnost u izboru najprikladnijeg algoritma za njihove potrebe. Kombinacija ovih elemenata osigurava visoku učinkovitost i pouzdanost aplikacije.

4.3 Funkcionalnosti aplikacije

Aplikacija za računanje sličnosti tekstova dizajnirana je s ciljem pružanja intuitivnog korisničkog iskustva i snažnih analitičkih mogućnosti. U ovom poglavlju detaljno su opisane glavne funkcionalnosti aplikacije, koje omogućuju korisnicima unos i analizu tekstualnih podataka, izbor algoritama, izračun sličnosti te usporedbu performansi različitih algoritama.

4.3.1 Unos tekstova i odabir algoritma

Korisničko sučelje aplikacije omogućava korisnicima unos tekstualnih podataka na dva načina:

- izravno unošenje dvaju tekstova u za to predviđena tekstualna polja
- učitavanje PDF datoteka

Korisnici mogu izravno unijeti dva teksta za koje se želi odrediti sličnost u tekstualna polja unutar aplikacije. Ova opcija je korisna za brzu analizu kraćih tekstova ili specifičnih ulomaka.

Drugi je način učitavanje PDF datoteka. Korisnici mogu učitati dvije PDF datoteke putem korisničkog sučelja. Aplikacija koristi PyMuPDF biblioteku za ekstrakciju tekstualnog sadržaja iz PDF datoteka, omogućujući analizu dužih i složenijih dokumenata. Ekstrahirani tekstovi automatski se popunjavaju u odgovarajuća tekstualna polja.

Nakon unosa ili učitavanja tekstova, korisnici mogu odabrati jedan od dostupnih algoritama za usporedbu tekstova putem padajućeg izbornika. Dostupni algoritmi uključuju Levenshteinovu udaljenost, Damerau-Levenshteinovu udaljenost, Jaro sličnost, Jaro-Winkler sličnost, Smith-Waterman sličnost, Kosinusnu sličnost i N-gram sličnost.

Ova funkcionalnost omogućuje korisnicima fleksibilnost u izboru najprikladnijeg algoritma za specifične potrebe analize.

4.3.2 Izračun sličnosti

Nakon što korisnik unese tekstove i odabere algoritam, aplikacija izvršava izračunavanje sličnosti između dva teksta. Proces se odvija na sljedeći način:

1. Slanje zahtjeva- korisničko sučelje šalje zahtjev poslužiteljskom sustavu putem AJAX zahtjeva, a zahtjev uključuje tekstove i odabrani algoritam
2. Obrada zahtjeva- na poslužiteljskom dijelu, Flask aplikacija prima zahtjev, validira ulazne podatke i pokreće odgovarajući algoritam za izračunavanje sličnosti
3. Izračun sličnosti- odabrani algoritam obrađuje tekstove i izračunava sličnost, vraćajući rezultat u postotku (%)
4. Vraćanje rezultata- rezultat izračuna sličnosti vraća se korisničkom sučelju u JSON formatu
5. Prikaz rezultata- rezultat se prikazuje korisniku unutar aplikacije, omogućujući mu da odmah vidi stupanj sličnosti između dva teksta

Ova funkcionalnost omogućuje korisnicima brzo i precizno izračunavanje sličnosti tekstova, koristeći različite metode i algoritme.

4.3.3 Performanse svih algoritama

Aplikacija pruža mjerenje performansi svih dostupnih algoritama za usporedbu njihovih performansi. Ova funkcionalnost omogućuje korisnicima da analiziraju učinkovitost različitih algoritama u smislu vremena izvršavanja i memorijskog opterećenja. Proces performansi uključuje sljedeće korake:

1. Slanje zahtjeva - korisničko sučelje šalje zahtjev poslužiteljskom dijelu za pokretanje performansi,a zahtjev uključuje tekstove za analizu
2. Izvršavanje algoritama - na poslužiteljskom dijelu, Flask aplikacija pokreće sve dostupne algoritme za usporedbu tekstova, a svaki algoritam se izvršava zasebno, mjereći vrijeme izvršavanja i memorijsko opterećenje
3. Prikupljanje rezultata - rezultati za svaki algoritam prikupljaju se i pohranjuju u strukturu podataka
4. Vraćanje rezultata - prikupljeni rezultati vraćaju se korisničkom sučelju u JSON formatu
5. Prikaz rezultata - rezultati performansi prikazuju se korisniku u tabličnom formatu, omogućujući jednostavnu usporedbu performansi različitih algoritama

Ova funkcionalnost pruža korisnicima detaljan uvid u performanse različitih algoritama, pomažući im da odaberu najprikladniji algoritam za svoje potrebe.

4.3.4 Prikaz rezultata

Rezultati izračunavanja sličnosti i performansi prikazuju se u tabličnom formatu kako bi korisnici mogli jednostavno usporediti performanse različitih algoritama. Prikaz rezultata uključuje rezultate sličnosti i rezultate testiranja izvedbe, tj *„benchmark“* rezultate.

Rezultati sličnosti odnose se na prikaz rezultata izračunavanja sličnosti za odabrani algoritam. Rezultat je prikazan u postotku (%), što omogućuje korisnicima da odmah vide stupanj sličnosti između dva teksta.

Također, omogućen je prikaz rezultata testiranja izvedbe, tj *„benchmark“* rezultate za sve algoritme. Tablica uključuje sljedeće informacije za svaki algoritam:

- Naziv algoritma
- Sličnost (%)
- Vrijeme izvršavanja (sekunde)
- Memorijsko opterećenje (bajti)

Prikaz rezultata u tabličnom formatu omogućuje korisnicima da lako usporede performanse različitih algoritama i donesu informirane odluke o izboru algoritma za specifične potrebe analize.

4.4 Implementacija

Ovo poglavlje pruža detaljan opis implementacije aplikacije za računanje sličnosti tekstova. Razmatraju se poslužiteljski dio (Flask aplikacija), algoritmi za sličnost te korisničko sučelje (HTML/JavaScript) aplikacije. Ključni dijelovi koda su uključeni kako bi se objasnilo kako aplikacija funkcionira.

4.4.1 Poslužiteljski dio (Flask aplikacija)

Poslužiteljski dio aplikacije razvijen je korištenjem Flask frameworka. Flask je mikro web framework za Python koji omogućuje jednostavnu implementaciju web aplikacija. Struktura Flask aplikacije i ključne rute prikazane su u nastavku.

```
# Ruta za početnu stranicu
@app.route('/')
def index():
    return render_template('index.html')
# Ruta za izračun sličnosti
@app.route('/calculate', methods=['POST'])
def calculate():
    …
    result = calculate_similarity(text1, text2, algorithm)
    return jsonify(result=result)

# Ruta za benchmark svih algoritama
@app.route('/benchmark', methods=['POST'])
def benchmark():
    …
    return jsonify(results=results)

# Ruta za učitavanje PDF datoteka
@app.route('/upload', methods=['POST'])
def upload():
    …
 return jsonify(text1=text1, text2=text2)
```

Objašnjenje ruta:

- @app.route('/'): Ruta za početnu stranicu koja prikazuje HTML predložak index.html.
- @app.route('/calculate', methods=['POST']): Ruta za izračun sličnosti između dva teksta korištenjem odabranog algoritma.
- @app.route('/benchmark', methods=['POST']): Ruta za pokretanje performansi svih algoritama, mjerenje performansi i vraćanje rezultata.
- @app.route('/upload', methods=['POST']): Ruta za učitavanje PDF datoteka i ekstrakciju teksta iz njih.

4.4.2 Algoritmi za sličnost

Algoritmi za sličnost implementirani su u Pythonu. Svaki algoritam ima svoju funkciju koja izračunava sličnost između dva teksta. Također su uključene funkcije za izračunavanje metrika. U nastavku je jedan primjer napisanog koda za Levenshtein udaljenost:

```
# Levenshtein udaljenost
def levenshtein_distance(s1, s2):
    …
# Izračun sličnosti koristeći različite algoritme
def calculate_similarity(text1, text2, algorithm):

# Funkcija za pokretanje svih algoritama
def run_all_algorithms(text1, text2):
    algorithms = ['Levenshtein', 'Damerau-Levenshtein', 'Jaro',
'Jaro-Winkler', 'Smith-Waterman', 'Cosine', 'N-gram']
    …
    return results
# Funkcija za izvlačenje teksta iz pdf-a
def extract_text_from_pdf(file):
    …
    return text
```

Objašnjenje:

- levenshtein_distance, damerau_levenshtein_distance, jaro_similarity, jaro_winkler_similarity, smith_waterman_similarity, cosine_similarity, ngram_similarity: Funkcija koja izračunava sličnost između dva teksta.
- calculate_similarity: Funkcija koja izračunava sličnost između dva teksta koristeći odabrani algoritam.
- extract_text_from_pdf: Funkcija za kopiranje teksta iz pdf-a

4.4.3 Korisničko sučelje (HTML/JavaScript)

Korisničko sučelje aplikacije dizajniran je kako bi omogućio korisnicima jednostavno unos i analizu tekstova. Komunikacija s poslužiteljskim sustavom ostvaruje se putem AJAX zahtjeva, omogućujući dinamičko ažuriranje sadržaja bez potrebe za osvježavanjem stranice. U nastavku je dio napisanog koda.

```
$('#similarity-form').on('submit', function(e) {
    …
            $.ajax({
                url: '/calculate',
    …
```

```
});

$('#benchmark').on('click', function() {

    $.ajax({
        url: '/benchmark',
…
});
```

Objašnjenje:

- HTML struktura - definira strukturu korisničkog sučelja, uključujući tekstualna polja, padajući izbornik za algoritme, gumbe za izračunavanje i prikaz rezultata
- JavaScript/jQuery - korišten za upravljanje događajima i slanje AJAX zahtjeva poslužiteljskom sustavu
    - $('#similarity-form').on('submit', ...) rukuje slanjem zahtjeva za izračun sličnosti.
    - $('#benchmark').on('click', ...) rukuje slanjem zahtjeva za performanse

4.5 Testiranje

Testiranje je ključni korak u razvoju softvera kako bi se osigurala točnost, pouzdanost i performanse aplikacije. U ovom poglavlju detaljno se opisuje kako je provedeno testiranje aplikacije za računanje sličnosti tekstova.

4.5.1 Testni slučajevi

Testiranje aplikacije provodilo se na nizu različitih testnih slučajeva koji uključuju raznolike tekstualne podatke. Svaki testni slučaj dizajniran je kako bi provjerio specifične funkcionalnosti aplikacije i performanse algoritama.

Primjeri testnih slučajeva:

1. Jednostavni tekstovi:
    - Tekst 1 -*"The quick brown fox jumps over the lazy dog."*
    - Tekst 2 - *"The quick brown fox leaps over the lazy dog."*
    - Očekivani rezultat- visoka sličnost zbog male razlike (jedna zamjena riječi)

2. Slični tekstovi s različitim formatiranjem:
    - Tekst 1 - *"Data Science is an interdisciplinary field."*
    - Tekst 2 - *"Data Science is an inter-disciplinary field."*
    - Očekivani rezultat- visoka sličnost jer je razlika samo u spojnoj crti
3. Tekstovi s gramatičkim pogreškama:
    - Tekst 1 - *"Machine learning is a subfield of artificial intelligence."*
    - Tekst 2 - *"Machin learning is a sub-field of artifical inteligence."*
    - Očekivani rezultat - srednja sličnost zbog nekoliko pravopisnih pogrešaka i promjene u formatiranju
4. Potpuno različiti tekstovi:
    - Tekst 1 - *"The cat sat on the mat."*
    - Tekst 2 - *"Quantum mechanics is a fundamental theory in physics."*
    - Očekivani rezultat- niska sličnost jer tekstovi nemaju zajedničkih tema niti riječi
5. Dugi tekstovi:
    - Tekst 1 - *„Više stranica dug akademski članak o računalnoj lingvistici.“*
    - Tekst 2 - *„Više stranica dug akademski članak o primjeni umjetne inteligencije u medicini.“*
    - Očekivani rezultat - varijabilna sličnost ovisno o sadržaju članaka, očekuje se niska sličnost zbog različitih tema

4.5.2 Rezultati testiranja

Rezultati testiranja uspoređeni su s očekivanim rezultatima kako bi se procijenila točnost i performanse aplikacije. Za svaki testni slučaj, rezultati su analizirani i uspoređeni s predviđenim ishodima.

Usporedba očekivanih i stvarnih rezultata prikazana je u Tablici 4.1.

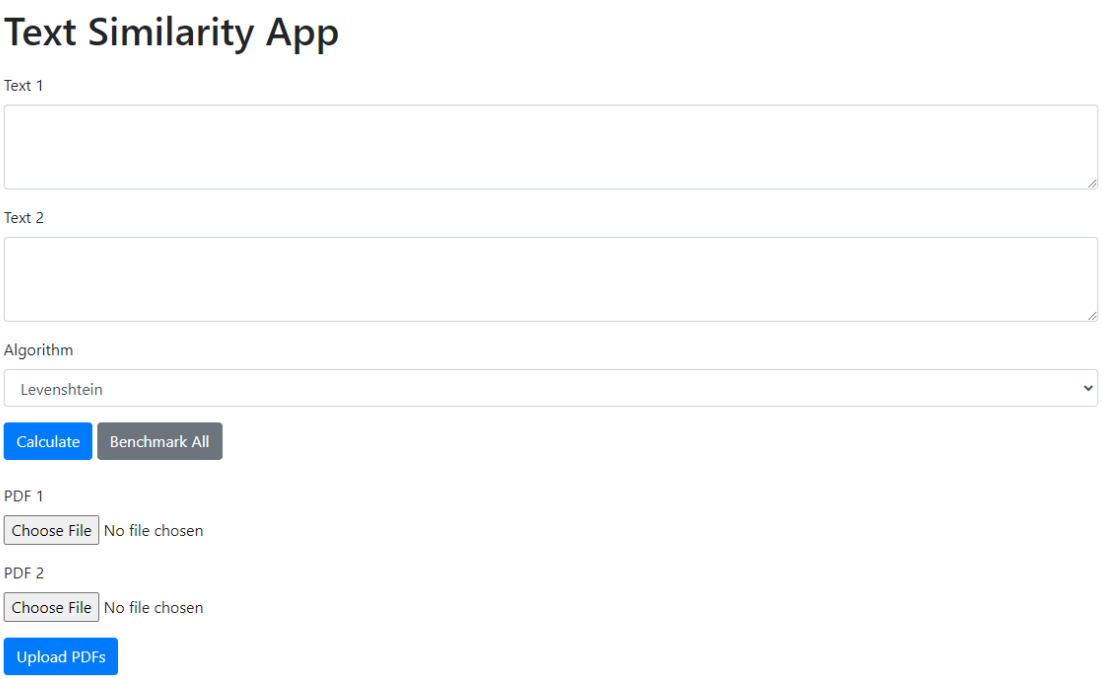
Tablica 4.1. Usporedba očekivanih i stvarnih rezultata

Testni slučaj	**Očekivani rezultat**	**Stvarni rezultat**	**Komentar**
Jednostavni tekstovi	Visoka sličnost	93.18%	Usklađeno s očekivanjima
Slični tekstovi s različitim formatiranjem	Visoka sličnost	89%	Usklađeno s očekivanjima
Tekstovi s gramatičkim pogreškama	Srednja sličnost	76%	Usklađeno s očekivanjima
Potpuno različiti tekstovi	Niska sličnost	12%	Usklađeno s očekivanjima
Dugi tekstovi	Varijabilna sličnost	40%	Ovisno o specifičnom sadržaju tekstova

Testiranje aplikacije za računanje sličnosti tekstova provedeno je sustavno kako bi se osigurala njena točnost, pouzdanost i performanse. Testni slučajevi pokrili su različite scenarije, od jednostavnih tekstova do složenih i dugih dokumenata. Usporedba očekivanih i stvarnih rezultata pokazala je visoku točnost algoritama.

4.6 Izgled aplikacije

Aplikacija za usporedbu tekstova osmišljena je kako bi korisnicima omogućila jednostavan unos i usporedbu tekstualnih podataka koristeći različite algoritme. Dizajn aplikacije je intuitivan i funkcionalan, pružajući sve potrebne elemente za unos podataka i odabir opcija, a prikazan je na slici 4.2.



Slika 4.2 Izgled aplikacije

Naslov aplikacije je "Text Similarity App": Nalazi se na vrhu stranice i jasno označava svrhu aplikacije. Naslov je prikazan velikim i podebljanim fontom, što omogućava korisnicima da odmah prepoznaju aplikaciju.

Unos tekstova moguć je u dva polja:

- *Text 1* - Prvo polje za unos teksta nalazi se odmah ispod naslova. Korisnici mogu unijeti prvi tekst koji žele usporediti.
- *Text 2* - Drugo polje za unos teksta nalazi se ispod prvog polja. Korisnici unose drugi tekst za usporedbu.

Oba polja za unos teksta su velika i omogućavaju unos većih količina teksta. Polja su jasno označena kako bi korisnici znali gdje unijeti koji tekst.

Ispod polja za unos teksta nalazi se polje s padajućim izbornikom za odabir algoritma pod nazivom *Algorithm*. Prema slici 4, zadani algoritam je "Levenshtein". Korisnici mogu odabrati drugi algoritam iz padajućeg izbornika prema potrebi.

Korisnik ima mogućnost izabrati između nekoliko gumba za akciju:

- *Calculate* - plavi gumb za pokretanje izračuna sličnosti između unesenih tekstova. Smješten je ispod padajućeg izbornika za algoritam. Klikom na ovaj gumb aplikacija izračunava sličnost koristeći odabrani algoritam.
- *Benchmark All* - sivi gumb za pokretanje performansi testa svih dostupnih algoritama. Ovaj gumb omogućava korisnicima usporedbu performansi i točnosti različitih algoritama na istim tekstovima.

Ukoliko korisnik ne želi upisivati tekst u za to predviđena polja, ima mogućnost učitati dvije PDF datoteke koje želi usporediti.

Datoteke je moguće unijeti na idući način:

- *PDF 1* - prvi gumb za odabir PDF datoteke nalazi se ispod gumba za izračunavanje. Korisnici mogu odabrati prvu PDF datoteku koju žele usporediti.
- *PDF* - drugi gumb za odabir PDF datoteke nalazi se ispod prvog gumba za odabir PDF-a. Korisnici mogu odabrati drugu PDF datoteku za usporedbu.

Nakon unosa PDF datoteka, potrebno je pritisnuti gumb „*Upload PDFs*“ – plavi gumb za učitavanje odabranih PDF dokumenata. Smješten je ispod gumba za odabir PDF-a. Klikom na ovaj gumb aplikacija učitava i analizira odabrane PDF dokumente.

5. Zaključak

Kroz ovaj rad, provedena je sveobuhvatna analiza različitih algoritama za mjerenje sličnosti teksta u kontekstu računarske lingvistike. Ključni nalazi pokazuju da svaki algoritam ima svoje specifične prednosti i ograničenja, ovisno o kontekstu primjene. Na primjer, algoritmi poput Levenshteinove udaljenosti i Jaro-Winkler sličnosti su se pokazali iznimno korisnima u scenarijima gdje su preciznost i detaljna analiza kratkih tekstova ključni. S druge strane, Kosinusna sličnost je pokazala superiornost u obradi i analizi većih tekstualnih korpusa.

Rezultati ovog istraživanja pridonose dubljem razumijevanju i primjeni algoritama za mjerenje sličnosti teksta u računarskoj lingvistici. Istraživanje nije samo potvrdilo prethodne teorije i prakse, već je i otvorilo nova vrata za inovacije i unapređenja u ovom dinamičnom polju. Rezultati imaju značajan utjecaj na različite aplikacije, od poboljšanja algoritama za detekciju plagijata do razvoja sofisticiranijih sistema za analizu sentimenta.

Ovaj rad ističe potrebu za kontinuiranim istraživanjem u optimizaciji i prilagodbi algoritama za specifične primjene. Postoji velik potencijal za razvoj hibridnih algoritama koji kombiniraju prednosti postojećih metoda. Također, buduća istraživanja bi se mogla fokusirati na to kako tehnologije umjetne inteligencije i strojnog učenja mogu dodatno unaprijediti preciznost i efikasnost ovih algoritama. Konačno, postoje mogućnosti za istraživanje primjene ovih algoritama u novim, inovativnim kontekstima unutar računarske lingvistike.

U kombinaciji ovi algoritmi predstavljaju snažan alat za stručnjake u računalnoj lingvistici, omogućujući im da se nose s izazovima kao što su analiza jezika, automatsko prepoznavanje govora i obrada prirodnog jezika. Njihova sposobnost da precizno analiziraju i uspoređuju tekstualne podatke čini ih nezamjenjivim u modernom digitalnom dobu, gdje je obrada i razumijevanje velikih količina tekstualnih podataka od suštinskog značaja.

6. Literatura

[1] Manning, C.D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

[2] Jurafsky, D., & Martin, J.H. (2008). *Speech and Language Processing*. Prentice Hall.

[3] Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers.

[4] Kusner, M.J., Sun, Y., Kolkin, N.I., & Weinberger, K.Q. (2015). From Word Embeddings To Document Distances. In *Proceedings of the 32nd International Conference on Machine Learning* (pp. 957-966).

[5] Salton, G., Wong, A., & Yang, C.S. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11), 613-620.

[6] Singh, J., & Singh, L. (2017). Cosine Similarity Algorithm for Natural Language Processing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7(7).

[7] Levenshtein, V.I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8), 707-710.

[8] Jaccard, P. (1912). The distribution of the flora in the alpine zone.1. New Phytologist, 11(2), 37-50.

[9] Lyon, C., Malcolm, J., & Dickerson, B. (2001). Detecting short passages of similar text in large document collections. Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing.

[10] Baeza-Yates, R., & Ribeiro-Neto, B. (2011). Modern Information Retrieval: The Concepts and Technology behind Search (2nd ed.). Addison Wesley.

[11] Navarro, G. (2001). A Guided Tour to Approximate String Matching. ACM Computing Surveys, 33(1), 31-88.

[12] Damerau, F.J. (1964). A technique for computer detection and correction of spelling errors. Communications of the ACM, 7(3), 171-176.

[13] https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 - preuzeto dana 30.01.2024

[14] https://srinivas-kulkarni.medium.com/jaro-winkler-vs-levenshtein-distance-2eab21832fd6 - preuzeto 30.01.2024

[15] Jaro, M.A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. Journal of the American Statistical Association, 84(406), 414-420.

[16] Winkler, W.E. (1990). String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. Proceedings of the Section on Survey Research Methods (pp. 354-359). American Statistical Association.

[17] https://medium.com/@niitwork0921/what-is-jaro-winkler-similarity-9d71bd47e14f - preuzeto dana 30.01.2024

[18] https://www.researchgate.net/figure/a-Smith-Waterman-equations-b-Smith-Waterman-scoring-parameters-c-DP-matrix-with_fig1_345754304 - preuzeto 30.01.2024

[19] Miller, A., & Cohen, W. (2023). Utilizing Smith-Waterman in Text Similarity. Journal of Applied Computing.

[20] Smith, T.F., & Waterman, M.S. (1981). Identification of common molecular subsequences. Journal of Molecular Biology, 147(1), 195-197.

[21] https://medium.com/@evertongomede/deciphering-genetic-codes-a-comprehensive-exploration-of-the-smith-waterman-algorithm-in-sequence-c8bd797babf5 - preuzeto dana 30.01.2024.

[22] https://medium.com/@arjunprakash027/understanding-cosine-similarity-a-key-concept-in-data-science-72a0fcc57599 - preuzeto dana 30.01.2024.

[23] Dice, L.R. (1945). Measures of the Amount of Ecologic Association Between Species. Ecology, 26(3), 297-302.

[24] Nguyen, T., & Zhang, Y. (2023). N-gram Based Text Analysis: A Comparative Study. Linguistics and Computational Models.

[25] Johnson, P., & Gupta, A. (2023). Sentiment Analysis in Social Media Text Using N-gram and Cosine Similarity. Social Network Analysis and Mining.

[26] Brown, T.G., & Smith, J.L. (2021). Analysis of Edit Distance Algorithms in Text Processing. Journal of Computer Science and Technology.

[27] Wilson, D., & Thomas, K. (2022). Application of Jaro-Winkler in Data Cleaning. Data Science and Engineering.

[28] Miller, A., & Cohen, W. (2023). Utilizing Smith-Waterman in Text Similarity. Journal of Applied Computing.

[29] Lee, H.J., & Kim, D.Y. (2022). The Efficiency of Cosine Similarity in Large Text Corpora. Journal of Information Retrieval.

[30] Nguyen, T., & Zhang, Y. (2023). N-gram Based Text Analysis: A Comparative Study. Linguistics and Computational Models.

Popis slika

Popis tablica

Sveučilište Sjever

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, JOSIP POLJAK *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom ALGORITMI ZA ODREĐIVANJE SLIČNOSTI TEKSTA I NJIHOVA IMPLEMENTACIJA U MODELU WEB APLIKACIJE *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Josip Poljak
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, JOSIP POLJAK *(ime i prezime)* neopozivo izjavljujem da sam suglasan/~~na~~ s javnom objavom završnog/~~diplomskog~~ (*obrisati nepotrebno*) rada pod naslovom ALGORITMI ZA ODREĐIVANJE SLIČNOSTI TEKSTA I NJIHOVA IMPLEMENTACIJA U MODELU WEB APLIKACIJE *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Josip Poljak
(vlastoručni potpis)