SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Domagoj Sabljić

PROCESIRANJE PRIRODNOG JEZIKA ZA RAZUMIJEVANJE POVRATNIH INFORMACIJA KLIJENATA U UGOSTITELJSKOJ INDUSTRIJI

PROJEKT

UVOD U UMJETNU INTELIGENCIJU

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Domagoj Sabljić

Matični broj: 0016153134

Studij: Informacijski i poslovni sustavi

PROCESIRANJE PRIRODNOG JEZIKA ZA RAZUMIJEVANJE POVRATNIH INFORMACIJA KLIJENATA U UGOSTITELJSKOJ INDUSTRIJI

PROJEKT

Mentor:

dr. sc. M. Schatten

Varaždin, siječanj 2024.

Domagoj Sabljić

Izjava o izvornosti

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Ovaj rad istražuje kako se napredne tehnike obrade prirodnog jezika mogu primijeniti za analizu, interpretaciju i odgovaranje na povratne informacije. Teorijsko-metodološko polazište rada obuhvaća analizu i implementaciju različitih modela strojnog i dubokog učenja, uključujući rekurentne neuronske mreže (RNN), vektorske prikaze riječi (engl. *word embeddings*), i sofisticiranije vrste rekurentnih neuronskih mreža.

Ključne riječi: Obrada prirodnog jezika, povratne informacije klijenata, ugostiteljstvo, strojno učenje, duboko učenje, RNN, GRU, LSTM, ekstrakcija n-grama.

Sadržaj

1. Uvod

U suvremenom poslovanju, posebice u ugostiteljskoj industriji, povratne informacije klijenata postaju temelj za unapređenje usluga i proizvoda, a sposobnost razumijevanja i adekvatnog reagiranja na povratne informacije postaje ključna za uspjeh. Razumijevanje ovih povratnih informacija, koje su često u tekstualnom obliku, predstavlja izazov zbog njihove subjektivnosti i raznolikosti izraza. U ovom kontekstu, primjena metoda obrade prirodnog jezika (engl. *Natural Language Processing* - NLP) otvara nove mogućnosti za automatsko analiziranje i razumijevanje ovih informacija.

Centralna uloga u ovom projektu pripada strojnom učenju, posebno tehnologijama dubokog učenja, koje omogućavaju računalima da iz velikih količina podataka *nauče* prepoznati kompleksne uzorke i nijanse u jeziku. Ovaj projekt istražuje kako se tehnike kao što su rekurentne neuronske mreže (RNN) i pred-trenirane vektorske reprezentacije riječi (engl. *word embeddings*) mogu koristiti za analizu i interpretaciju povratnih informacija klijenata. Te metode ne samo da omogućuju razumijevanje osnovnog značenja teksta, već i prepoznavanje suptilnih emocija i stavova, čime pružaju dublji uvid u iskustva i očekivanja klijenata.

Osim toga, projekt će istražiti i primjenu tehnika poput ekstrakcije najpopularnijih n-grama, koje mogu pomoći u identificiranju ključnih tema i trendova unutar povratnih informacija. Ovime se ne samo povećava efikasnost obrade velikih količina podataka, već se i omogućava bolje razumijevanje specifičnih aspekata usluge koji su važni za klijente.

Projekt postavlja temelje za detaljnije istraživanje navedenih tehnika i njihovu primjenu u stvarnom svijetu, čime se naglašava važnost NLP-a u unapređenju kvalitete usluge i zadovoljstva klijenata u ugostiteljskoj industriji.

2. Metode i tehnike rada

U razradi teme projekta "Procesiranje prirodnog jezika za razumijevanje povratnih informacija klijenata u ugostiteljskoj industriji", primijenjene su različite metode i tehnike kako bi se postigla maksimalna efikasnost i preciznost u obradi i analizi podataka.

Python: Kao osnovni programski jezik, odabran je Python zbog svoje svestranosti, čitljivosti i široko prihvaćenog statusa u *data science/ML* zajednici. Snažna podrška za biblioteke vezane za obradu podataka, statističku analizu i strojno učenje čini Python idealnim za ovakvu vrstu projekta.

TensorFlow: Ovaj programski okvir (engl. *framework*) za strojno učenje, razvijen od strane Google Brain tima, koristi se za izgradnju i treniranje modela dubokog učenja. TensorFlow nudi fleksibilnost u dizajniranju složenih arhitektura mreža, kao što su rekurentne neuronske mreže (RNN), što je ključno za obradu i analizu prirodnog jezika.

Kaggle skup podataka: Za potrebe treninga i validacije modela, korišteni su otvoreni skupovi podataka dostupne na Kaggle platformi.Kaggle nudi pristup velikom broju skupova podataka, što uključuje i one specifične za ugostiteljsku industriju.

Istraživačke Aktivnosti: Tijekom projekta, istraživačke aktivnosti su uključivale prikupljanje i obradu podataka, izgradnju i treniranje modela strojnog učenja te njihovu validaciju. Početna faza je uključivala analizu i pretprocesiranje prikupljenih podataka, nakon čega je uslijedilo modeliranje pomoću TensorFlow-a.

3. Konceptualni okvir

Ovaj dio obuhvaća temeljne definicije i teorijske okvire koji podupiru obradu prirodnog jezika. Detaljno se objašnjavaju koncepti kao što su rekurentne neuronske mreže, vektorski prikazi riječi, te se daje pregled različitih vrsta rekurentnih neuronskih mreža kao što su GRU i LSTM mreže, ističući njihovu ulogu i značaj u kontekstu analize povratnih informacija.

3.1. Rekurentne neuronske mreže

Razumijevanje sekvencijalnih podataka kao što su tekstovi ili vremenski nizovi ključno je za učinkovitu analizu i interpretaciju prirodnog jezika. Za ovakve vrste podataka, tradicionalne neuronske mreže često nisu dovoljne, budući da im nedostaje sposobnost povezivanja informacija kroz vrijeme. Upravo ovdje dolaze do izražaja rekurentne neuronske mreže (RNN - *Recurrent Neural Networks*), koje su specifično dizajnirane za rad sa sekvencijalnim podacima.

Za razliku od standardnih neuronskih mreža koje obrađuju ulazne podatke u jednom koraku, RNN-ovi imaju sposobnost 'pamćenja' prethodnih informacija kroz posebne izlazne vektore koje nazivamo skrivena stanja. To znači da kada RNN obrađuje novi element sekvence, on uzima u obzir i informacije koje su prikupljene iz prethodnih koraka uz trenutni ulaz odnosno aktivacije ne putuju samo iz smjera ulaznog do izlaznog sloja, već one dolaze i iz prošlog vremenskog koraka. Zbog toga je RNN arhitektura prikladna za analizu vremenskih nizova, gdje je kontekstualna ovisnost između uzastopnih točaka podataka od velike važnosti.

U svakom trenutku *t* RNN može imati dva izlaza, skriveno stanje i opcionalni izlaz za trenutak *t*. Skriveno stanje $h^{<t>}$ dobiva se kao funkcija prethodnog skrivenog stanja $h^{<t-1>}$ i trenutnog ulaza $x^{<t>}$. Za danu ulaznu sekvencu $x = (x^{<1>}, x^{<2>}, ..., x^{<T_x>})$ RNN podešava skriveno stanje h na sljedeći način.

$$h^{<t>} = \begin{cases} 0, & t = 0 \\ f(h^{<t-1>}, x^{<t>}), & t \geq 0 \end{cases}$$

gdje je f ne-linearna aktivacijska funkcija kao kompozicija logističke i linearne funkcije ili hiperbolička tanges funkcija [1].

Detaljnije to možemo razložiti na sljedeći način:

$$h^{<t>} = g_1\left(W_{hh}h^{<t-1>} + W_{hx}x^{<t>} + b_h\right)$$

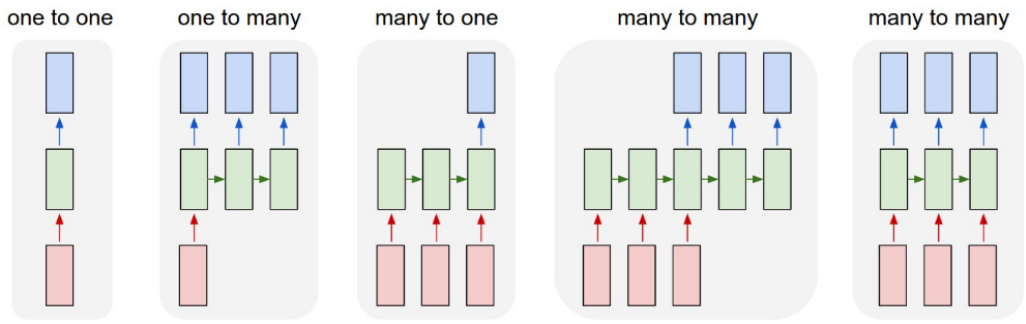
Opcionalno, izlaz u trenutku *t* dobivamo na sljedeći način:

$$\hat{y}^{<t>} = g_2\left(W_{yh}h^{<t>} + b_y\right)$$

Kod običnih rekurentnih neuronskih mreža g_1 je glatka, omeđena funkcija (npr. logistička funkcija), a odabir funkcije g_2 zavisi od vrste zadatka. Za binarnu klasifikaciju bi odabrali logističku funkciju dok bi za više-klasnu klasifikaciju odabrali *softmax*. W_hh, W_hx i W_yh su matrice težina, zajedničke za cijelu rekurentnu neuronsku mrežu, koje učimo algoritmom *unazadne propagacije kroz vrijeme*.

3.1.1. Konfiguracije rekurentnih neuronskih mreža

Rekurentne neuronske mreže (RNN) su iznimno prilagodljive i mogu se primjenjivati u različitim konfiguracijama ovisno o specifičnom zadatku i vrsti podataka. Ove različite konfiguracije omogućuju RNN-ovima da rješavaju širok spektar problema obrade prirodnog jezika, generiranja teksta, prepoznavanja govora i drugim primjenama.



Slika 1: RNN konfiguracije; preuzeto iz [2]

Na slici 1 imamo redom sljedeće konfiguracije rekurentni neuronskih mreža:

One-to-One: Ovdje se RNN ponaša slično kao tradicionalna, potpuno povezana (engl. *fully connected*) neuronska mreža. Ova konfiguracija se koristi u situacijama gdje nema potrebe za obradom sekvencijalnih podataka, već gdje je fokus na pojedinačnim, izoliranim instancama ulaza.

One-to-Many: Ovdje mreža prima jedan ulazni podatak i generira sekvencu izlaza. Primjeri uključuju generiranje glazbe gdje ulaz može biti opis, a izlaz niz nota. Još jedan primjer bi bio klasifikacija slika (engl. *image clasisification*) gdje je ulaz slika, a izlaz niz riječi (tekst) koji predstavlja opis slike.

Many-to-One: U ovom pristupu, mreža prima sekvencu ulaznih podataka i generira jedan izlaz. Ovo je uobičajeno u zadacima kao što su analiza sentimenta, gdje se na osnovu reče-

nice ili komentara određuje sentiment (pozitivan, negativan, neutralan).

Many-to-Many: Ovdje razlikujemo dva slučaja. U prvom slučaju duljina ulazne sekvence nije jednaka duljini izlazne sekvence i ta se konfiguracija primjenjuje kod strojnog prevođenja (rečenica prevedena sa hrvatskog na francuski ne mora biti jednake duljine). Drugi slučaj je kada su ulazna i izlazna sekvenca jednake duljine. Primjer toga bi prepoznavanje imena u tekstu (engl. *Named Entity Recognition - NER*) gdje je ulaz tekst, a izlaz sekvenca koja klasificira koji dio teksta predstavlja ime.

3.1.2. Dvosmjerne rekurentne neuronske mreže

Dvosmjerne rekurentne neuronske mreže (engl. *Bidirectional Recurrent Neural Networks - BiRNNs*) su vrsta rekurentnih neuronskih mreža koje se koriste za obradu sekvencijalnih podataka. Ove mreže su posebno korisne u zadacima kao što je prepoznavanje imenovanih entiteta u tekstu (engl. *Named Entity Recognition (NER)*), gdje je važno razumjeti kontekstualne veze između riječi koja se trenutno obrađuje te prethodnih i sljedećih riječi iz sekvence.

Na primjer, u rečenicama 'He said, "**Teddy** bears are on sale!"' i 'He said, "**Teddy** Roosevelt was a great President!"' [3], možemo jasno uočiti u kojem kontekstu se riječ 'Teddy' koristi kao ime. Jednosmjerna rekurentna neuronska mreža ima značajno ograničenje u ovom zadatku jer bi klasifikacija riječi 'Teddy' ovisila samo o prethodnim riječima u rečenici. Kako bismo prevladali ovu ograničenost, ključno je koristiti dvosmjernu rekurentnu neuronsku mrežu.

Osnovna ideja iza dvosmjernih rekurentnih neuronskih mreža je da se informacije obrađuju u dvije faze, unaprijed i unatrag kroz sekvencu. Intuitivno, kombiniramo dvije rekurentne neuronske mreže u jednu od kojih će jedna mreža ići naprijed kroz sekvencu, a druga unazad. Tako će izlaz u trenutku t ovisiti o dva skrivena stanja (iz $t-1$ i $t+1$), što nam daje alat za izgradnju moćnijih i robustnijih modela.

3.1.3. Problem iščezavajućih ili eksplodirajućih gradijenata

Problem iščezavajućih (engl. *vanishing*) ili eksplodirajućih (engl. *exploding*) gradijenata je ključni izazov u treniranju dubokih neuronskih mreža (posebno rekurentnih neuronskih mreža), a javlja se tijekom procesa propagacije unazad.

Iščezavajući gradijenti se javljaju kada vrijednosti gradijenata postaju vrlo male tijekom unazadne propagacije. To se obično događa u dubokim mrežama gdje se gradijent mora propagirati kroz mnoge slojeve. Kao rezultat, proces učenja se odvija jako sporo ili se uopće ne mogu naučiti značajke povezane s ranijim slojevima, zbog beznačajnog podešavanja odgovarajućih težina, što otežava hvatanje dugoročnih ovisnosti u podacima koje je posebno važno kod sekvecijalnih podataka.

S druge strane, eksplodirajući gradijenti se javljaju kada vrijednosti gradijenata postanu prevelike tijekom unazadne propagacije, te se rjeđe javljaju u kontekstu rekurentnih neuronskih mreža [1].

Ove pojave otežavaju zadatak optimizacijskih metoda temeljenih na gradijentnom spustu, ne samo zbog varijacije u veličini gradijenata, već i zato što je utjecaj dugoročnih ovisnosti skriven iza utjecaja kratkoročnih [1]. Za rješavanje ovih problema, postoji nekoliko opcija:

- **Obrezivanje gradijenata (engl. *gradient clipping*):** Ograničavanje veličine gradijenata na određenu maksimalnu vrijednost kako bi se spriječili eksplodirajući gradijenti.

- **Promjena aktivacijskih funkcija:** Korištenje ReLU ili njegovih varijanti , koje nemaju zasićenje na pozitivnom dijelu, može pomoći u ublažavanju problema iščezavajućih gradijenata.

- **Korištenje LSTM ili GRU jedinica:** Ove naprednije strukture RNN-a primjenjuju mehanizme 'vrata', odnosno sklopova (engl. *gates*) koji omogućavaju bolje upravljanje protokom informacija i gradijenata, čime se smanjuje rizik od iščezavajućih gradijenata.

U sljedećim sekcijama posvetit ćemo više pažnje LSTM i GRU mrežama kao rješenjima navedenih problema.

3.2. *Long short-term memory (LSTM)* mreže

Prvi pokušaj razvoja sofisticiranije rekurentne jedinice rezultirao je razvojem LSTM jedinice 1997. godine (Hochreiter and Schmidhuber). Glavna karakterizacija ove jedinice je upotreba mehanizma sklopova. Svaka LSTM jedinica je konstruirana od tri sklopa, sklopa za zaboravljanje, ulaznog i izlaznog sklopa, koji zajednički upravljaju protokom informacija u i iz internog stanja jedinice. LSTM može po potrebi brisati i dodavati informacije u internom stanju, a te operacije reguliraju odgovarajući sklopovi. Ključna ideja je ta da interno stanje funkcionira kao pokretna traka koja prolazi kroz sve jedinice mreže i olakšava da informacija ide po toj traci nepromijenjena [4].

LSTM prvo odlučuje koje dijelove internog stanja će odbaciti (zaboraviti) preko sklopa za zaboravljanje koji gleda trenutni ulaz, skriveno i interno stanje iz prethodnog trenutka, te vraća vrijednosti između 0 i 1 za svaku vrijednost iz prethodnog internog stanja.

$$f^{<t>} = \sigma(W_f x^{<t>} + U_f h^{<t-1>} + V_f c^{<t-1>})$$

Zatim moramo odlučiti koje nove informacije ćemo pohraniti u interno stanje. Sklop za ulaz odlučuje koje informacije treba ažurirati te se kreira novi vektor $\tilde{c}$ koji sadrži kandidate za dodavanje u interno stanje.

$$i^{<t>} = \sigma(W_i x^{<t>} + U_i h^{<t-1>} + V_i c^{<t-1>})$$

$$\tilde{c}^{<t>} = \tanh(W_c x^{<t>} + U_c h^{<t-1>})$$

Sada trebamo ažurirati interno stanje iz prethodnog trenutka, a to ćemo uraditi tako što pomnožimo staro interno stanje sa sklopom za zaboravljanje i tome dodamo rezultat množenja ulaznog sklopa i vektora kandidata za interno stanje (kandidatne vrijednosti skalirane prema tome koliko želimo ažurirati svaku vrijednost internog stanja.

$$c^{<t>} = f^{<t>} c^{<t-1>} + i^{<t>} \tilde{c}^{<t>}$$

Na kraju odlučujemo što će biti izlazne vrijednosti. Ta odluka je odgovornost izlaznog sklopa koji će biti pomnožen sa internim stanjem i tako selektirati one dijelove za koje je odlučeno da će biti dio izlaza.

$$o^{<t>} = \sigma(W_o x^{<t>} + U_o h^{<t-1>} + V_o c^{<t>})$$

$$h^{<t>} = o^{<t>} \tanh c^{<t>}$$

Sada smo vidjeli kako uz pomoć sklopova LSTM mreža, za razliku od obične rekurentne neuronske mreže, ima sposbnost odlučivanja o zadržavanju i odbacivanju informacija. Ako LSTM na početku ulazne sekvence detektira važnu značajku onda tu značajku može lagano zapamtiti na dulje vrijeme i time postići hvaćanje dugoročnih veza [1].

3.3. *Gated recurrent (GRU)* mreže

"Nasljednik" LSTM jedinice je RNN jedinica zvana *gated recurrent unit* koja je zasnovana na sličnim principima uz male promjene koje mogu biti interpretirane kao pojednostavljenja LSTM jednice. Ova pojednostavljenja rezultiraju lakšom implementacijom uz manje vektorskih operacija unutar same jedinice što za posljedicu ima manju iskorištenost resursa i efikasnost.

Sastavni dijelovi GRU jedinice su sklop za ažuriranje (engl. *update gate*) i sklop za ponovno postavljanje odnosno *resetiranje* (engl. *reset gate*) koji u principu odlučuju koja informacija će ići dalje u sljedeću jednicu, odnosno koje informacije treba zapamtiti. Ovi sklopovi mogu biti istrenirani tako da pamte informacije relevantne za konačni izlaz, a nebitne informacije ignoriraju. Kada dobijemo ulaz u trenutku *t* prvo računamo vrijednost sklopa za ažuriranje ($z^{<t>}$) na sljedeći način:

$$z^{<t>} = \sigma(W_z x^{<t>} + U_z h^{<t-1>})$$

Logistička funkcija (σ) će vratiti rezultate u rasponu od 0 do 1. Ovaj sklop pomaže modelu u određivanju kolika količina informacija iz prethodnih $t-1$ koraka treba biti proslijeđena dalje, odnosno zapamćena.

Zatim se računa vrijednost sklopa za ponovno postavljanje ($r^{<t>}$)koja služi za određivanje količine prošlih informacija koju trebamo zaboraviti.

$$r^{<t>} = \sigma(W_r x^{<t>} + U_r h^{<t-1>})$$

Sljedeći korak je računanje kandidatnog skrivenog stanja (engl. *candidate activation*):

$$\tilde{h}^{<t>} = \tanh(Wx^{<t>} + U(r^{<t>} \odot h^{<t-1>}))$$

Kada je vrijednost sklopa za ponovno postavljanje blizu nule, tada taj sklop djeluje na jedinicu tako da se ona ponaša kao da čita prvi ulaz sekvence, dopuštajući joj da ignorira ili zaboravi prethodno stanje.

Da bi ilustrirali rad ovog sklopa uzmimo primjer analize sentimenta. Ako imamo komentar (npr. "*Ovaj proizvod sam kupio prije par dana i ... Sve u svemu, nisam prezadovoljan.*") koji nakon nekoliko paragrafa teksta sadrži ključnu informaciju za klasifikaciju sentimenta onda će mreža naučiti postaviti $r^{<t>} \approx 0$, kako bi se više fokusirali na recentnije informacije [5].

Aktivacija $h^{<t>}$ se dobiva iz svih prethodno izračunatih komponenti:

$$h^{<t>} = (1 - z^{<t>})h^{<t>} + z^{<t>}\tilde{h}^{<t>}$$

Ovdje vidimo kako sklop $z^{<t>}$ odlučuje koliko se informacije prenosi dalje [1].

3.4. Vektorski prikazi riječi (engl. *word embeddings*)

Kako bismo mogli poslati tekst rekurentnim neuronskim mrežama na način koji im je razumljiv, moramo prvo konvertirati tekst u numerički oblik. Svaku riječ u tekstu treba prikazati pomoću zasebnog vektora koji jedinstveno identificira svaku riječ iz vokabulara. Vektorski prikazi riječi (engl. *word embeddings*) su ključni koncept u području obrade prirodnog jezika zbog njihove sposobnosti predstavljanja riječi kao višedimenzionalnih vektora u prostoru. Tradicionalni načini reprezentacije riječi, kao što je "one-hot encoding," (nul-vektor duljine jednake duljini vokabulara s jednicom na indeksu dane riječi) ne zadržavaju bogatstvo semantičkog značenja riječi. Rezultat ovog pristupa su reprezentacije riječi koje su međusobno potpuno neovisne (euklidska udaljenost između svake dvije riječi je jednaka).

S druge strane, vektorski prikazi riječi rješavaju ovaj problem pridružujući svakoj riječi vektor u višedimenzionalnom prostoru. Ovi vektori se treniraju na velikim količinama teksta, što omogućuje modelima da nauče semantičke odnose među riječima. Na primjer, slične riječi će imati slične vektorske reprezentacije, a vektori će uhvatiti odnose kao što su sinonimi, antonimi ili čak kontekstualne veze među riječima. Ovaj pristup omogućuje NLP sustavima da bolje razumiju prirodni jezik i izvode složenije zadatke.

Možemo treinrati vlastite vektorske prikaze na prikupljenom tekstu ili preuzeti dostupne predtrenirane vektorske prikaze kao što su GloVe ili Word2Vec. Ovi prikazi su trenirani na ogromnim količinama teksta i sadrže općenite informacije o jeziku. Kod zadataka u kojima ne-

mamo veliku količinu podataka predtrenirani vektorski prikazi mogu unaprijediti generalizaciju modela, odnosno pomoći u izbjegavanju preprilagođenosti (engl. *overfit*). Također imamo i opciju da težine (engl. *weights* preuzetih prikaza nastavimo podešavati tijekom treniranja našeg modela, što nije preporučljivo za male količine podataka [6].

3.4.1. Analogijsko zaključivanje

Koncept vektorskih prikaza riječi možemo intuitivno prikazati kroz njihovo svojstvo analogijskog zaključivanja. Uzmimo sljedeće riječi i njihove odgovarajuće vektore:

Značajka/Riječ	**Muškarac**	**Žena**	**Kralj**	**Kraljica**
Spol	-1	1	-0.95	0.97
Hrana	0.002	0.001	0.002	0.003

Tablica 1: Implicitne analogije u kontinuiranim prostornim reprezentacijama riječi; prema [7]

Vidimo da u ovom slučaju vektori prikazuju dvije značajke riječi (spol te da li dana riječ predstavlja hranu). Vektorski prikazi su u praksi dimenzijski bogatiji te nemaju lako interpretabilne značajke. Sada možemo postaviti sljedeće pitanje: 'Muškarac je ženi isto što i kralj ?'. Označimo vektore za pojedinu riječ sa e_{rijec}, te izračunajmo razliku parova riječi iz tablice:

$$e_{muskarac} - e_{zena} \approx \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

$$e_{kralj} - e_{kraljica} \approx \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Iz ovoga vidimo da je glavna značajka po kojoj se dani parovi riječi razlikuju spol. Jedan način na koji možemo provesti analogijsko zaključivanje je taj da probamo naći riječ w koja zadovoljava jednakost:

$$e_{muskarac} - e_{zena} \approx e_{kralj} - e_w$$

Iz toga slijedi:

$$e_w \approx e_{kralj} - e_{muskarac} + e_{zena}$$

Drugim riječima, moramo pronaći riječ w koja maksimizira funkciju sličnosti $sim(e_w, e_{kralj} - e_{muskarac} + e_{zena})$ [7]. Za funkciju sim najčešće se koristi kosinusna sličnost (ako je kut između dva vektora 0 funkcija poprima maksimalnu vrijednost koja je izvedena iz formule skalarnog produkta:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos(\theta)$$

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

Pa možemo zapisati:

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

Ovime smo dobili način za mjerenje sličnosti između dvije dane riječi preko njihovih vektorskih prikaza.

3.4.2. GloVe vektori (engl. *Global Vectors*)

Dvije glavne porodice učenja vektorskih prikaza riječi se zasnivaju na globalnoj faktorizaciji matrice (kao što je *Latent Semantic Analysis* - LSA) i tehnikama lokalnog konteksta kao što je *skip-gram* model. Ove tehnike dolaze s nekim manama i nedostatcima.

Metode poput LSA efikasno koriste statističke informacije, ali se slabije snalaze na zadatku analogije riječi, ukazujući na suboptimalnu strukturu vektorskog prostora. Metode poput *skip-grama* su bolje u zadatku analogijskog zaključivanja, ali ne iskorištavaju statistiku korpusa jer se oslanjaju na odvojene lokalne kontekstualne prozore umjesto na globalne statistiku korpusa.

Glavni problem kod tehnika matrične faktorizacije je taj da riječi s najviše ponavljanja previše utječu na mjeru sličnosti između parova riječi. Za razliku od metoda matrične faktorizacije, metode bazirane na kontekstualnim prozorima imaju nedostatak što ne djeluju izravno na statistici matrice supojavljivanja riječi korpusa. Umjesto toga, ovi modeli skeniraju kontekstualne prozore kroz cijeli korpus, što ne koristi količinu ponavljanja u podacima.

Kao odgovor na ove nedostatke, uvodi se GloVe, koji koristi globalnu statistiku riječi korpusa i lokalni kontekst. GloVe modelira vektorske reprezentacije riječi koristeći težinski model najmanjih kvadrata (engl. *weighted least squares model*) koji tijekom učenja uzima u obzir broj supojavljivanja riječi u korpusu te time maksimalno iskorištava dostupne statističke informacije. GloVe proizvodi vektorski prostor riječi s značajnom podstrukturom, pokazujući vrhunske rezultate na zadacima analogijskog zaključivanja i sličnosti riječi, kao i na testovima prepoznavanja

imenovanih entiteta (NER) [8].

4. Opis implementacije

U ovom dijelu opisuje se skup podataka te implementacija komponenti programskog rješenja kao i implementacija prethodno spomenutih modela. Klasifikator za analizu sentimenta će biti implementiran u Pythonu u obliku sekvencijalnog modela pomoću TensorFlow modula. Cjelovito rješenje dostupno je u GitHub repozitoriju.

4.1. Eksplorativna analiza i vizualizacija podataka

Prije početka s izradom rješenja, odnosno treniranja sekvencijalnih modela, moramo se upoznati sa skupom podataka na kojem ćemo ove modele trenirati. Za ovaj problem iskoristit ćemo *Trip Advisor Hotel Reviews* skup podataka dostupan na Kaggle platformi.

Učitat ćemo skup u `pandas DataFrame` i prikazati prvih nekoliko redova.

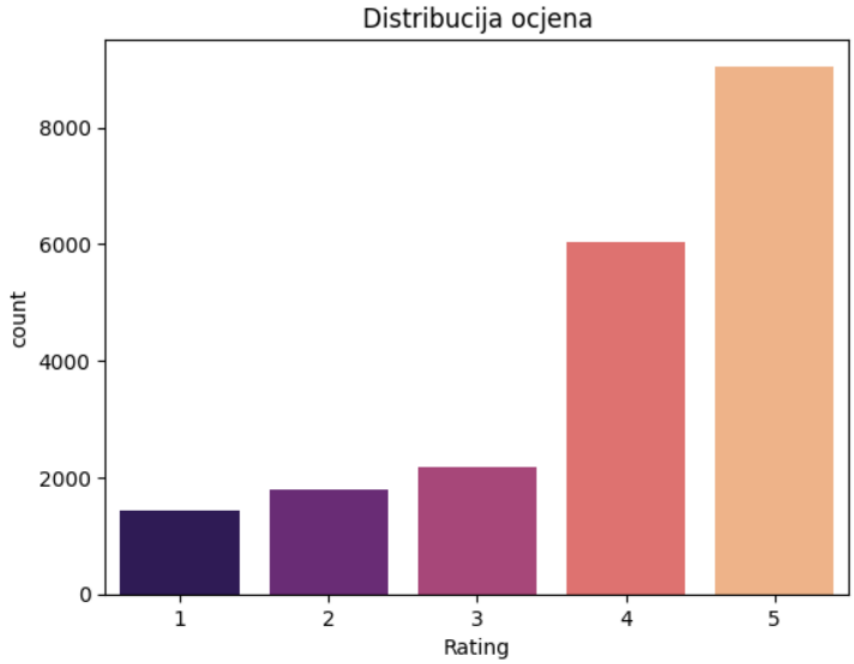
```
1 import pandas as pd

2 df = pd.read_csv('./data/tripadvisor_hotel_reviews.csv')
3 df.head()
```

Isječak koda 1: Učitavanje skupa podataka u `DataFrame df`

	Review	Rating
0	nice hotel expensive parking got good deal stay hotel anniversary, arrived late evening took adv...	4
1	ok nothing special charge diamond member hilton decided chain shot 20th anniversary seattle, sta...	2
2	nice rooms not 4* experience hotel monaco seattle good hotel n't 4* level.positives large bathro...	3
3	unique, great stay, wonderful time hotel monaco, location excellent short stroll main downtown s...	5
4	great stay great stay, went seahawk game awesome, downfall view building did n't complain, room ...	5

Slika 2: Prikaz prvih redova `DataFrame`-a

U ovom skupu podataka imamo recenzije klijenata u tekstualnom obliku i pripadajuće ocjene (1-5). Ocjene nisu uniformno distribuirane što se može vidjeti sa sljedeće slike:



Slika 3: Distribucija ocjena

Vidimo da je najviše pozitivnih ocjena (4 i 5) dok su ostale ocjene slablje zastupljene u skupu.

Zatim čistimo recenzije iz skupa podataka tako što uklanjamo stop-riječi (engl. *stop-words*) i specijalne znakove kako bi mogli nastaviti s analizom. Specijalne znakove ćemo ukloniti pomoću regularnih izraza, a stop-riječi uklanjamo tako da za svaku riječ u recenziji provjerimo je li ta riječ u popisu stop-riječi iz `nltk.corpus` modula i ako je onda je ignoriramo. Ovo sve radimo u funkciji `clean` koju ćemo kasnije primjeniti na sve recenzije u skupu koristeći `pandas` metodu `apply`.

```
1 import re
2 from nltk.corpus import stopwords
3 nltk.download('stopwords')

4 def clean(review):

5     review = review.lower()
6     review = re.sub('[^a-z A-Z 0-9-]+', '', review)
7     review = " ".join([word for word in review.split() if word not in
      ↪  stopwords.words('english')])

8     return review

9 df['Review'] = df['Review'].apply(clean)
```

Isječak koda 2: Čišćenje recenzija

Jedan način za dobivanje uvida u recenzije je taj da izlistamo najzastupljenije riječi. Kako bi to mogli, prvo moramo recenzije razložiti na zasebne riječi koje ćemo zatim staviti u zaseban stupac `Review_lists`.

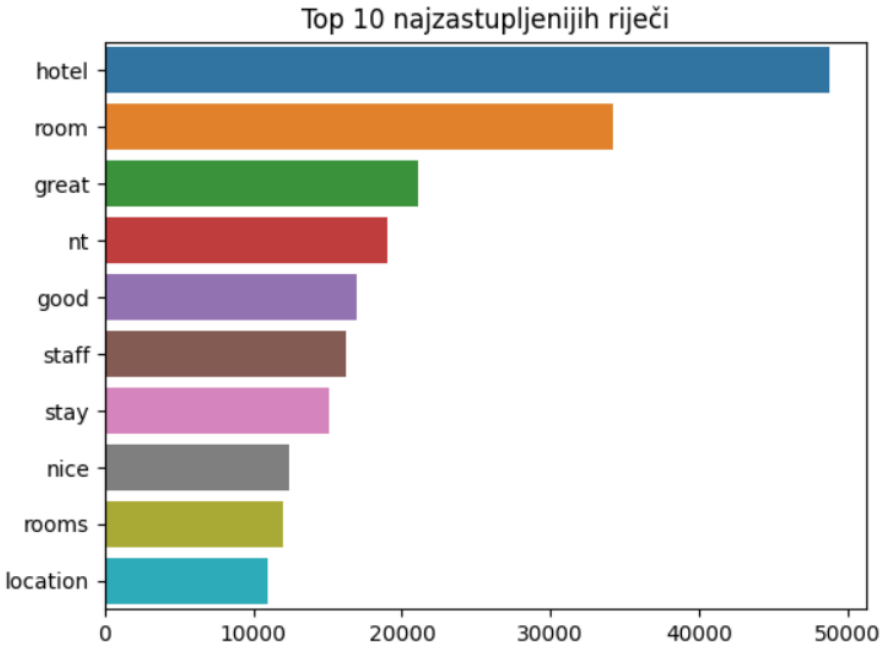
```
def split_into_words(text):
    text_list = text.split()
    return text_list

df['Review_lists'] = df['Review'].apply(split_into_words)
```

Isječak koda 3: Razlaganje recenzija na zasebne riječi

Nakon toga možemo stvoriti korpus koji će sadržavati sve riječi iz recenzija tako da iteriramo preko svih lista u stupcu `Review_lists` te preko sadržaja pojedinačne liste i taj sadržaj (riječi) dodamo u listi `corpus`. Sada možemo izvući najzastupljenije rijječi pomoću strukture podataka `Counter`. Ova struktura će automatski kreirati uređene parove oblika (riječ, frekvencija). Pozivom metode `most_common` u varijablu `most_common_words` pohranjujemo 10 najzastupljenijih riječi. Kako bi razdvojili riječi i frekvencije u odvojene liste iskoristit ćemo funkciju `zip`, te ćemo rezultat (liste `words` i `freq`) iskoristiti za vizualizaciju frekvencija riječi stupčastim dijagramamom.

```
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns

corpus = [word for review_list in df['Review_lists'] for word in review_list]
most_common_words = Counter(corpus).most_common(10)

words, freq = zip(*most_common_words)

sns.barplot(x=list(freq), y=list(words))
plt.title('Top 10 najčešće upotrebljenih riječi')
plt.show()
```

Isječak koda 4: Dohvaćanje i vizualizacija najzastupljenijih riječi



Slika 4: Najzastupljenije riječi u korpusu

Da bismo dobili dublji uvid u recenzije, možemo iskoristiti *n*-grame. *N*-grami su sekvence od *n* riječi izvučene iz teksta, koje nam omogućuju da analiziramo i shvatimo kontekstualne obrasce i frekvencije pojavljivanja određenih riječi ili fraza unutar recenzija.

```
from sklearn.feature_extraction.text import CountVectorizer

def display_ngram(n, df):
    cv = CountVectorizer(ngram_range=(n,n))
    n_gram = cv.fit_transform(df['Review'])
    count_values = n_gram.toarray().sum(axis=0)
    ngram_df = pd.DataFrame(sorted([(count_values[i], k) for k, i in
    ↪  cv.vocabulary_.items()], reverse = True))
    ngram_df.columns = ["frequency", "ngram"]

sns.barplot(x=ngram_df['frequency'][:10], y=ngram_df['ngram'][:10])
plt.title('Top 10 najčešćih n-grama')
plt.show()
```

Isječak koda 5: Dohvaćanje i vizualizacija najzastupljenijih *n*-grama

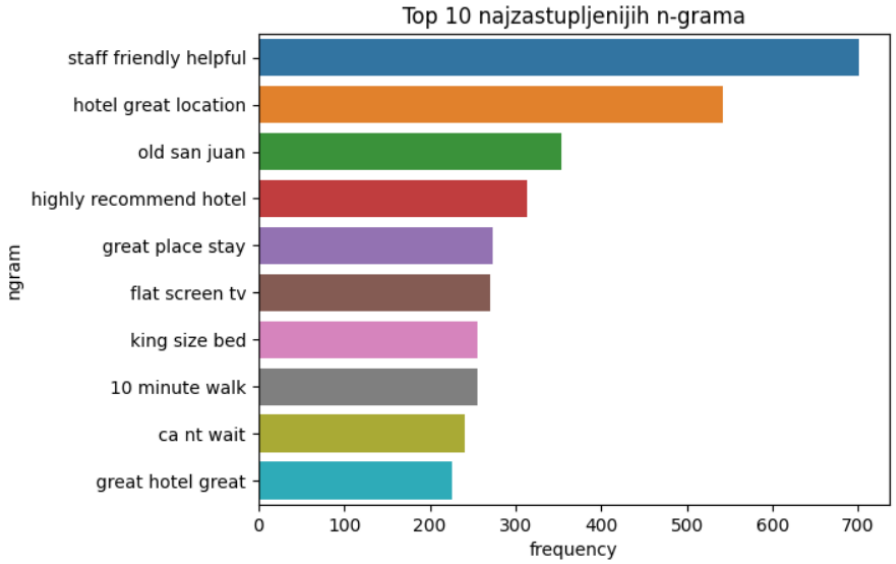
U isječku koda 5 imamo `CountVectorizer`, alat koji se koristi za pretvaranje teksta u vektorski format, što je standardni korak u obradi prirodnog jezika. Specifično, pretvara tekst u vektor frekvencija pojavljivanja riječi ili n-grama. Funkcija `display_ngram` uzima dva parametra, `n` koji određuje veličinu n-grama (npr. 1 za unigrame, 2 za bigrame) i `pandas DataFrame df`.

Unutar funkcije, `cv` se inicijalizira kao instanca `CountVectorizer`-a, gdje je `ngram_range` postavljen na (n, n) što znači da će se generirati *n*-grami točno veličine *n*. Zatim se koristi metoda `fit_transform` na stupcu 'Review' `DataFrame`-a kako bi prilagodili model podacima u stupcu, te transformirali tekstove recenzija u numerički format (točnije, u matricu frekvencija

n-grama).

`toarray().sum(axis=0)` sumira frekvencije svakog n-grama preko svih recenzija. Rezultat je niz ukupnih frekvencija za svaki n-gram. Nakon toga se kreira novi DataFrame koji sadrži frekvencije i same n-grame. Sortiranje se vrši po frekvenciji u silaznom poretku.

Naposljetku, koristi se `sns.barplot` za prikaz stupčastog dijagrama prvih 10 najčešćih n-grama.



Slika 5: Najzastupljeniji *n*-grami u korpusu

Iz slike 5 možemo izvući korisne informacije o usluzi hotela. Da na primjer imamo veliku količinu recenzija koju smo prikupili sa straince nekog hotela i iz unutarnjih izvora, mogli bi zaključiti što klijenti posebno zapažaju kod usluge, bilo da je pozitivno ili negativno, te bi hotel na osnovu izvučenih informacija mogao nastaviti s dobrim stvarima i popraviti loše. U našem primjeru sa slike vidimo da su najzstupljeniji *n*-grami prilično pozitivni, što ne začuđuje s obzirom na to da smo već naglasili kako je većina recenzija pozitivna (ocjene 4 i 5, slika 3).

4.2. Analiza sentimenta sekvencijalnim modelima

Već smo opisali kako funkcioniraju neki sekvencijalni modeli kao što su rekurentna neuronsta mreža, LSTM i GRU mreža. Sada ćemo upotrijebiti te modele kako bi kreirali sustav za analizu sentimenta koji recenzije svrstava u tri kategorije (negativna, neutralna i pozitivna recenzija). Kao što je već spomenuto za implementaciju modela koristit ćemo TensorFlow. Prije treniranja i izgradnje modela moramo pretprocesirati i pripremiti podatke.

Prvo što ćemo napraviti je konvertiranje ocjene iz `df['Rating']` stupca u tri vrijednosti odnosno klase (0 - negativno, 1 - neutralno i 2 - pozitivno).

```
df['Label'] = df['Rating'].apply(lambda x: 0 if x in [1, 2] else (1 if x == 3 else
↪  2))
```

Isječak koda 6: Konvertiranje ocjene u klase

U isječku koda 6 gledamo vrijednosti u `df['Rating']` stupcu i na njega primjenjujemo `lambda` izraz koji će vratiti 0 ako je trenutna ocjena 1 ili 2, 1 ako je ocjena 3 i 2 u svim ostalim slučajevima (ocjene 4 i 5). Na kraju dobivene klase spremamo u stupac `Label`.

Sada kada imamo odgovarajuće oznake možemo podijeliti skup podataka na skup za treniranje, validaciju i testiranje. Za to koristimo funkciju `train_test_split` iz `sklearn` modula. Odvojit ćemo 20% skupa za testiranje, 20% za validaciju (25% od preostalih 80%) i ostalih 60% za treniranje.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df["Review"], df["Label"],
↪  test_size=0.2, random_state=72)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25,
↪  random_state=72)
```

Isječak koda 7: Podijela originalnog skupa podataka na skup za treniranje, validaciju i testiranje

Sljedeći korak je pretprocesiranje recenzija. Za taj zadatak kreirana je funkcija prikazana u isječku koda 8 (`custom_preprocessing_sw_removal`). Glavni cilj ove funkcije je transformacija recenzija na način da se očuvaju ključni elementi koji utječu na sentiment, dok se uklanjaju manje bitni dijelovi. `SnowballStemmer` se koristi za svođenje riječi na njihov korijenski oblik, što pomaže u standardizaciji riječi kako bi se poboljšao performans modela [9] (npr., *running* se svodi na *run*).

Zatim se definira skup stop-riječi (riječi koje se često pojavljuju, ali obično nose malo semantičkog značenja, poput *and*, *the*, *in*) i iz njega se izuzimaju neke ključne riječi koje su važne za analizu sentimenta, poput *not*, *no*, *nt*, *very*. Ovo je važno jer uklanjanje ovih riječi može promijeniti značenje rečenice, posebno u kontekstu sentimenta.

`custom_preprocessing_sw_removal` je funkcija dizajnirana za prilagođeno pretprocesi-

ranje teksta. Ako ulazni tekst nije u `string` formatu, konvertira se u `string`. Ovo osigurava da funkcija može obraditi različite tipove ulaznih podataka. Nakon toga, tekst se pretvara u mala slova kako bi se osigurala konzistentnost, posebno jer se riječi u prirodnom jeziku mogu pojaviti u različitim oblicima (npr., *House* i *house*). Interpunkcijski znakovi se uklanjaju jer obično ne doprinose analizi sentimenta, a mogu otežati procesiranje teksta. Na kraju, tekst se dijeli na riječi, vrši se korjenovanje svake riječi, i uklanjaju se sve stop-riječi koje nisu označene kao važne za zadržavanje.

```
from nltk.stem.snowball import SnowballStemmer

stop_words = set(stopwords.words('english'))
words_to_keep = {'not', 'no', 'nt', 'very'}
filtered_stop_words = stop_words - words_to_keep

stemmer = SnowballStemmer("english")

def custom_preprocessing_sw_removal(text):
    # Provjera je li tekst string i pretvaranje u string ako nije
    if not isinstance(text, str):
        text = str(text)

    # Pretvaranje teksta u mala slova
    text = text.lower()
    # Uklanjanje interpunkcije
    text = re.sub(r'[^\w\s]', '', text)
    # Stemming i uklanjanje stopwordsa osim onih koji utječu na sentiment
    text = " ".join(stemmer.stem(word) for word in text.split() if word not in
    ↪  filtered_stop_words)

    return text
```

Isječak koda 8: Funkcija za pretprocesiranje recenzija

Funkciju `custom_preprocessing_sw_removal` sada moramo primjeniti na naše skupove podataka.

```
X_train = X_train.apply(custom_preprocessing_sw_removal)
X_val = X_val.apply(custom_preprocessing_sw_removal)
X_test = X_test.apply(custom_preprocessing_sw_removal)
```

Isječak koda 9: Primjena pretprocesiranja na skupove za treniranje, validaciju i testiranje

U isječku koda 10 možemo vidjeti kako se podaci pripremaju za efikasno treniranje, validaciju i testiranje modela.

Parametar `batch_size` određuje broj uzoraka koji će se obraditi zajedno u jednoj iteraciji. U ovom slučaju, model će u svakoj iteraciji treniranja uzeti 32 uzorka iz skupa podataka.

`tf.data.Dataset.from_tensor_slices` je funkcija koja stvara TensorFlow Dataset objekte iz zadanog skupa podataka. Dataset objekti su efikasni za obradu velikih količina podataka i mogu se koristiti za dohvat podataka u grupama ili serijama.

Metoda `shuffle` se koristi za nasumično miješanje elemenata u skupu podataka. `buffer_size`,

u ovom slučaju `len(X_train)`, određuje broj elemenata u skupu podataka koji će biti pomiješani. To osigurava da nasumičan redoslijed podataka nasumičan što je korisno za izbjegavanje pristranosti tijekom treniranja. Nakon toga, podaci se grupiraju u serije veličine 32. `prefetch` se koristi za optimizaciju učitavanja podataka, omogućavajući modelu da asinkrono dohvaća serije podataka tijekom procesa treniranja, što poboljšava efikasnost i smanjuje vrijeme čekanja.

Priprema skupova podataka za validaciju (`val_ds`) i testiranje (`test_ds`) odvija se na sličan način. Međutim, za ove skupove podataka se ne koristi miješanje.

```
batch_size = 32

train_ds = tf.data.Dataset.from_tensor_slices((X_train,
↪  y_train)).shuffle(len(X_train)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
val_ds = tf.data.Dataset.from_tensor_slices((X_val,
↪  y_val)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_ds = tf.data.Dataset.from_tensor_slices((X_test,
↪  y_test)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

Isječak koda 10: Konfiguracija skupova podataka

U sljedećem koraku `VOCAB_TOKENS` varijablom postavljamo broj tokena (riječi) koje `TextVectorization` sloj treba zadržati u svom vokabularu. Sirovi tekst mora biti obrađen kako bi ga preveli u oblik s kojim model može raditi. U tu svrhu koristi se `TextVectorization` sloj iz `tf.keras.layers` modula. Stvara se instanca ovog sloja koji služi za pretvaranje teksta u numerički oblik. Postavljanjem `max_tokens` na 5000, ograničava se vokabular sloja na najčešćih 5000 riječi.

Zatim je potrebno prilagoditi (engl.*fit*) sloj na skup podataka sa metodom `adapt`. Metoda `map` se koristi za izvlačenje samo tekstualnih podataka iz skupa podataka (bez oznaka), jer nam za `TextVectorization` sloj trebaju samo recenzije.

```
VOCAB_TOKENS = 5000

encoder = tf.keras.layers.TextVectorization(max_tokens=VOCAB_TOKENS)
encoder.adapt(train_ds.map(lambda text, labels: text))
```

Isječak koda 11: Priprema enkodera

4.2.1. Implementacija obične rekurentne neuronske mreže

Prvi model koji ćemo razmatrati je obična rekurentna neuronska mreža. Ključna komponenta TensorFlow-a koja znatno olakšava izgradnju modela je Keras API. Keras API nudi intuitivno sučelje visoke razine za izradu modela dubokog učenja.

Prije izgradnje modela trebamo uvesti potrebne module. Prvo uključujemo `tensorflow` pod aliasom `tf`. Za izgradnju modela treba nam `Sequential` modul koji će poslužiti kao struktura koja nam omogućava *slaganje* slojeva modela jedan na drugi. `Adam` je popularni algoritam optimizacije koji se koristi u treniranju neuronskih mreža. `EarlyStopping` je vrsta povratnog poziva (*callback*) koji se koristi za prekid treniranja modela kada određeni uvjet (npr. validacijska točnost) više ne pokazuje poboljšanje, što pomaže u sprječavanju preprilagođenosti modela (engl. *overfit*). Na kraju, iz `tf.keras.layers` uvozimo potrebne slojeve.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import (Embedding, SimpleRNN, GRU,
                                     LSTM, Bidirectional, Dense, Dropout)
```

Isječak koda 12: Uključivanje potrebnih modula

Sada možemo definirati naš prvi model. Za prvi sloj postavljamo ranije definirani `encoder` kako bi podaci bili pretvoreni u željeni format (niz indeksa tokena). Zatim slijedi `Embedding` sloj koji pretvara nizove indeksa riječi u nizove vektora. Svaka riječ u vokabularu dobiva svoj vektor, a ti vektori se podešavaju tijekom procesa učenja. Riječi sličnog značenja često završavaju sličnim vektorima nakon treniranja. `input_dim` je veličina vokabulara, a `output_dim` je dimenzija vektora (u ovom slučaju 64). `mask_zero=True` omogućava sloju da rukuje sekvencama različitih duljina. `SimpleRNN` je sloj jednostavne rekurentne neuronske mreže s 32 neurona u svakoj RNN jedinici. Prvi `Dense` sloj ima 16 neurona i koristi aktivacijsku funkciju `ReLU` (engl. *Rectified Linear Unit*). Ovaj sloj je potpuno povezan (engl. *fully connected*) i služi za daljnju obradu značajki izvučenih iz RNN sloja. `Dropout` je sloj koji nasumično "isključuje" 50% neurona tijekom treniranja te tako pomaže u smanjenju preprilagođenosti modela. Posljednji `Dense` sloj s 3 neurona i aktivacijskom funkcijom `softmax` koristi se za klasifikaciju. Ovaj sloj pretvara izlaz iz prethodnih slojeva u vjerojatnosti za svaku od 3 ciljane klase.

U ovom primjeru smo se odlučili za jednosmjernu rekurentnu neuronsku mrežu, ali smo jednostavno, uz pomoć `tf.keras.layers.Bidirectional` *wrapper*-a, mogli kreirati i dvosmjerni RNN (`tf.keras.layers.Bidirectional(SimpleRNN(32))`.

```
rnn_model = Sequential([
        encoder,
        Embedding(input_dim=len(encoder.get_vocabulary()), output_dim=50,
        ↪  mask_zero=True),
        SimpleRNN(32),
        Dense(16, activation='relu'),
        Dropout(0.5),
        Dense(3, activation='softmax')
])
```

Isječak koda 13: Definiranje obične rekurentne neuronske mreže

Nakon što definiramo model moramo ga kompajlirati koristeći metodu `compile` kojoj kao argumente šaljemo željenu optimizacijsku metodu (`Adam` sa stopom učenja (engl. *learning rate*), 0.0003 u ovom primjeru), funkciju gubitka (engl. *loss function*) te metriku praćenja. Pozivom `fit` metode na kompiliranom modelu pokrećemo proces treniranja. Ovoj metodi šaljemo skup podataka treniranje sa oznakama (ciljnim vrijednostima), broj epoha (koliko puta *prođemo* kroz skup podataka), skup na kojem će se model validirati tijekom procesa treniranja, te posljednji argument `callbacks` gdje stavljamo posebne funkcije koje se mogu izvršiti tijekom procesa treniranja. U ovom slučaju u listu koju šaljemo postavljamo samo `EarlyStopping` koji prekida treniranje modela prije nego što dosegne maksimalni broj epoha ako se zadani `monitor` parametar (u ovom slučaju `val_accuracy`, točnost na validacijskom skupu) ne poboljšava tijekom određenog broja epoha (`patience=3`). To znači da će treniranje prestati ako se točnost na validacijskom skupu ne poboljša tijekom 3 uzastopne epohe. Na kraju povijest treniranja spremamo u varijablu `history` koju se može iskoristiti za vizualizaciju točnosti (engl. *accuracy*) i gubitka (engl. *loss*) kroz epohe.

```
rnn_model.compile(optimizer=Adam(learning_rate=3e-4),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = rnn_model.fit(train_ds, epochs=10, validation_data=val_ds,
↪  callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,
↪  monitor="val_accuracy")])
```

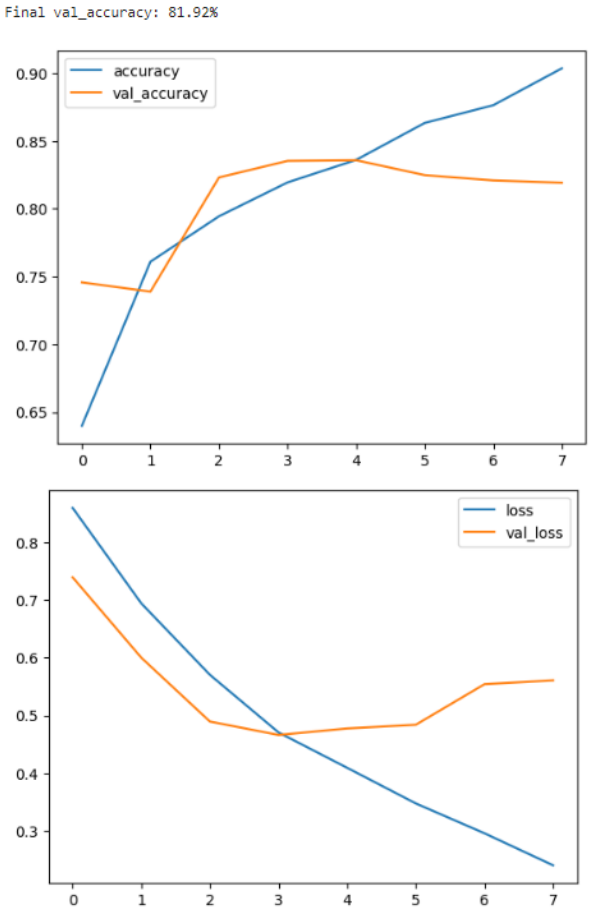
Isječak koda 14: Kompajliranje RNN modela

Nakon što je proces treniranja dovršen možemo pogledati rezultate odnosno završnu točnost na validacijskom modelu te krivulje učenja.

Na slici 7 vidimo da je završna točnost modela na validacijskom skupu 81.92% te se po krivuljama učenja vidi kako kako točnost kroz vrijeme raste, a gubitak pada. Pred kraj treniranja vidimo kako je gubitak na validacijskom setu počeo rasti pa je treniranje zaustavljeno (`EarlyStopping`). Sa grafova možemo zaključiti da se model počeo preprilagođavati skupu za treniranje. To vidimo i iz slike, gdje iz zadnjeg retka čitamo kako je točnost na skupu za treniranje bila 90.35%, a na skupu za validaciju već spomenutih 81.92%.

```
Epoch 1/10
385/385 [==============================] - 310s 792ms/step - loss: 0.8606 - accuracy: 0.6399 - val_loss: 0.7400 - val_accuracy: 0.7457
Epoch 2/10
385/385 [==============================] - 270s 702ms/step - loss: 0.6947 - accuracy: 0.7609 - val_loss: 0.6004 - val_accuracy: 0.7389
Epoch 3/10
385/385 [==============================] - 257s 667ms/step - loss: 0.5706 - accuracy: 0.7945 - val_loss: 0.4894 - val_accuracy: 0.8231
Epoch 4/10
385/385 [==============================] - 252s 653ms/step - loss: 0.4705 - accuracy: 0.8194 - val_loss: 0.4661 - val_accuracy: 0.8353
Epoch 5/10
385/385 [==============================] - 255s 661ms/step - loss: 0.4090 - accuracy: 0.8360 - val_loss: 0.4774 - val_accuracy: 0.8358
Epoch 6/10
385/385 [==============================] - 254s 661ms/step - loss: 0.3472 - accuracy: 0.8633 - val_loss: 0.4839 - val_accuracy: 0.8248
Epoch 7/10
385/385 [==============================] - 256s 667ms/step - loss: 0.2953 - accuracy: 0.8764 - val_loss: 0.5543 - val_accuracy: 0.8209
Epoch 8/10
385/385 [==============================] - 260s 676ms/step - loss: 0.2397 - accuracy: 0.9035 - val_loss: 0.5609 - val_accuracy: 0.8192
```

Slika 6: Treniranje obične rekurentne neuronske mreže kroz epohe



Slika 7: Rezultati treniranja obične rekurentne neuronske mreže

Pozivom metode `evaluate` možemo evalurati naš model na skupu za testiranje. Na slici 8 vidimo da model na skupu za testiranje ima sličan performans kao i na validacijskom skupu (81.04%).

```
test_loss, test_acc = rnn_model.evaluate(test_ds)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

Isječak koda 15: Evaluacija RNN modela

```
129/129 [==============================] - 8s 63ms/step - loss: 0.5715 - accuracy: 0.8104
Test Loss: 0.5714691281318665
Test Accuracy: 0.8104415535926819
```

Slika 8: Rezultati evaluacije za RNN

4.2.2. Implementacija GRU mreže

GRU mrežu u TensorFlow-u definiramo na isti način kao i običnu rekurentnu neuronsku mrežu. Jedina razlika je korištenje `GRU` sloja umjesto `SimpleRNN` sloja i promjena u vrijednostima parametara za treniranje (engl. *hyperparameters*).

```
gru_model = Sequential([
        encoder,
        Embedding(len(encoder.get_vocabulary()), 50, mask_zero = True),
        GRU(32),
        Dense(32, activation='relu',
        ↪  kernel_regularizer=tf.keras.regularizers.l1(0.005)),
        Dropout(0.5),
        Dense(3, activation='softmax')
])

gru_model.compile(optimizer=Adam(learning_rate=1e-3),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = gru_model.fit(train_ds, epochs=10, validation_data=val_ds,
↪  callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,
↪  monitor="val_accuracy")])
```

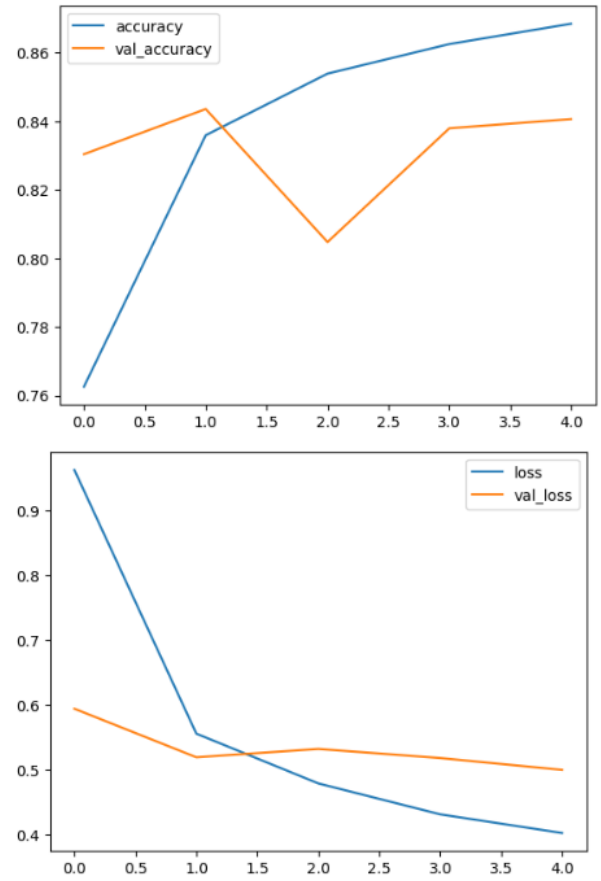
Isječak koda 16: Definicija i treniranje GRU mreže

U daljnjim primjerima se koriste samo dvosmjerne mreže (za LSTM i primjer implementacije sa GloVe vektorima), a sve ostale implementacije su dostupne u ranije navedenom repozitoriju.

```
Epoch 1/10
385/385 [==============================] - 43s 96ms/step - loss: 0.9631 - accuracy: 0.7626 - val_loss: 0.5947 - val_accuracy: 0.8304
Epoch 2/10
385/385 [==============================] - 21s 54ms/step - loss: 0.5562 - accuracy: 0.8359 - val_loss: 0.5198 - val_accuracy: 0.8436
Epoch 3/10
385/385 [==============================] - 16s 41ms/step - loss: 0.4795 - accuracy: 0.8539 - val_loss: 0.5327 - val_accuracy: 0.8048
Epoch 4/10
385/385 [==============================] - 14s 37ms/step - loss: 0.4318 - accuracy: 0.8625 - val_loss: 0.5186 - val_accuracy: 0.8380
Epoch 5/10
385/385 [==============================] - 13s 33ms/step - loss: 0.4030 - accuracy: 0.8685 - val_loss: 0.5003 - val_accuracy: 0.8407
```

Slika 9: Treniranje GRU mreže kroz epohe

```
Final val_accuracy: 84.07%
```



Slika 10: Rezultati treniranja GRU mreže

```
129/129 [==============================] - 2s 13ms/step - loss: 0.5098 - accuracy: 0.8385
Test Loss: 0.5097711086273193
Test Accuracy: 0.8384972214698792
```

Slika 11: Rezultati evaluacije za GRU

Iz rezultata vidimo da se GRU model dobro generalizira na neviđene podatke (84.07% i 83.58% točnosti na validacijskom i testnom skupu).

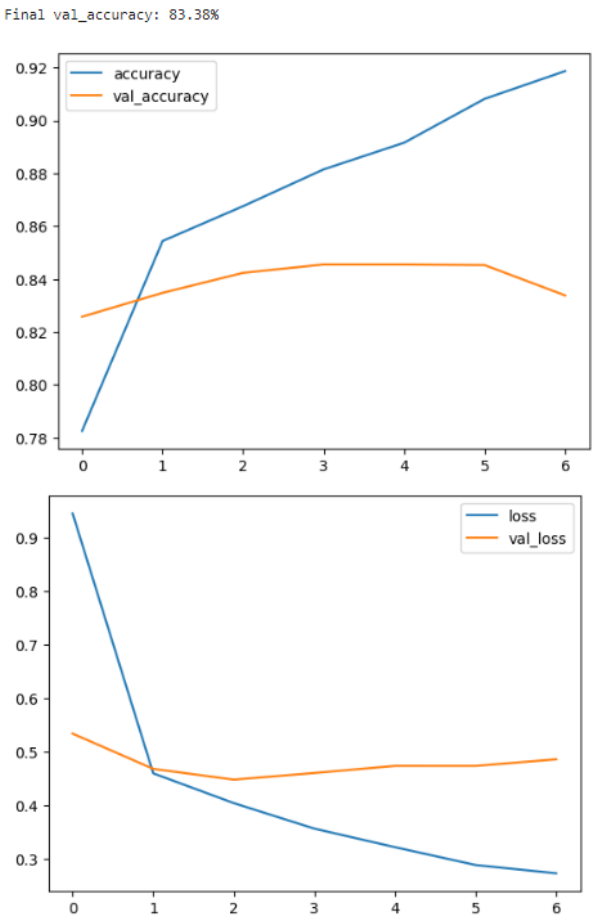
Još ćemo razmotriti dvosmjerni GRU model.

```
bi_gru_model = tf.keras.models.Sequential([
        encoder,
        Embedding(len(encoder.get_vocabulary()), 50, mask_zero = True),
        Bidirectional(GRU(32)),
        Dense(32, activation='relu',
        ↪  kernel_regularizer=tf.keras.regularizers.l1(0.005)),
        Dropout(0.4),
        Dense(3, activation='softmax')
])

bi_gru_model.compile(optimizer=Adam(learning_rate=1e-3),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = bi_gru_model.fit(train_ds, epochs=10, validation_data=val_ds,
↪  callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,
↪  monitor="val_accuracy")])
```

Isječak koda 17: Definicija i treniranje dvosmjerne GRU mreže

```
Epoch 1/10
385/385 [==============================] - 54s 116ms/step - loss: 0.9447 - accuracy: 0.7826 - val_loss: 0.5341 - val_accuracy: 0.8258
Epoch 2/10
385/385 [==============================] - 27s 70ms/step - loss: 0.4600 - accuracy: 0.8544 - val_loss: 0.4683 - val_accuracy: 0.8348
Epoch 3/10
385/385 [==============================] - 22s 57ms/step - loss: 0.4044 - accuracy: 0.8676 - val_loss: 0.4482 - val_accuracy: 0.8424
Epoch 4/10
385/385 [==============================] - 21s 54ms/step - loss: 0.3568 - accuracy: 0.8815 - val_loss: 0.4606 - val_accuracy: 0.8455
Epoch 5/10
385/385 [==============================] - 19s 49ms/step - loss: 0.3222 - accuracy: 0.8916 - val_loss: 0.4739 - val_accuracy: 0.8455
Epoch 6/10
385/385 [==============================] - 19s 48ms/step - loss: 0.2890 - accuracy: 0.9082 - val_loss: 0.4738 - val_accuracy: 0.8453
Epoch 7/10
385/385 [==============================] - 17s 44ms/step - loss: 0.2735 - accuracy: 0.9187 - val_loss: 0.4861 - val_accuracy: 0.8338
```

Slika 12: Treniranje dvosmjerne GRU mreže kroz epohe



Slika 13: Rezultati treniranja dvosmjerne GRU mreže

Iz rezultata vidimo da je performans dvosmjernog GRU modela nešto slabija od njegovog prethodnika (83.38% i 82.90% točnosti na validacijskom i testnom skupu).

4.2.3. Implementacija LSTM mreže

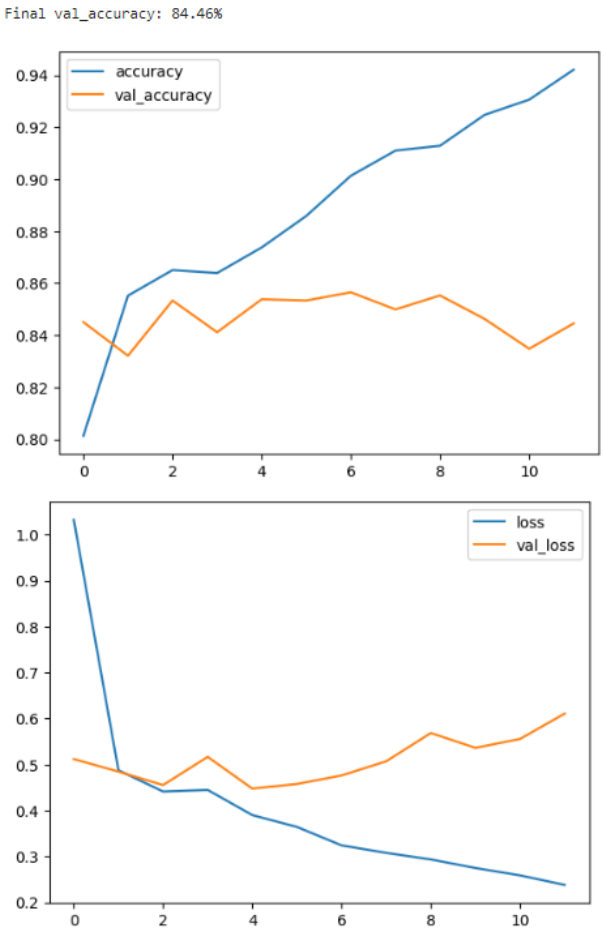
Isto kao i u prethodnom primjeru, definiramo `Sequential` strukturu uz promjenu vrste jedinice sekvencijalnog modela.

```
bi_lstm_model = tf.keras.models.Sequential([
        encoder,
        Embedding(len(encoder.get_vocabulary()), 50, mask_zero = True),
        Bidirectional(LSTM(64)),
        Dense(32, activation='relu',
        ↪  kernel_regularizer=tf.keras.regularizers.l1(0.005)),
        Dropout(0.5),
        Dense(3, activation='softmax')
])

bi_lstm_model.compile(optimizer=Adam(learning_rate=1e-3),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = bi_lstm_model.fit(train_ds, epochs=20, validation_data=val_ds,
↪  callbacks=[tf.keras.callbacks.EarlyStopping(patience=5,
↪  monitor="val_accuracy")])
```

Isječak koda 18: Definicija i treniranje dvosmjerne LSTM mreže

```
Epoch 1/20
385/385 [==============================] - 56s 124ms/step - loss: 1.0329 - accuracy: 0.8013 - val_loss: 0.5119 - val_accuracy: 0.8456
Epoch 2/20
385/385 [==============================] - 27s 71ms/step - loss: 0.4876 - accuracy: 0.8552 - val_loss: 0.4845 - val_accuracy: 0.8321
Epoch 3/20
385/385 [==============================] - 24s 61ms/step - loss: 0.4412 - accuracy: 0.8651 - val_loss: 0.4552 - val_accuracy: 0.8533
Epoch 4/20
385/385 [==============================] - 21s 55ms/step - loss: 0.4447 - accuracy: 0.8639 - val_loss: 0.5167 - val_accuracy: 0.8411
Epoch 5/20
385/385 [==============================] - 19s 49ms/step - loss: 0.3899 - accuracy: 0.8738 - val_loss: 0.4476 - val_accuracy: 0.8538
Epoch 6/20
385/385 [==============================] - 19s 48ms/step - loss: 0.3641 - accuracy: 0.8859 - val_loss: 0.4576 - val_accuracy: 0.8533
Epoch 7/20
385/385 [==============================] - 17s 45ms/step - loss: 0.3240 - accuracy: 0.9013 - val_loss: 0.4762 - val_accuracy: 0.8565
Epoch 8/20
385/385 [==============================] - 18s 46ms/step - loss: 0.3077 - accuracy: 0.9110 - val_loss: 0.5070 - val_accuracy: 0.8499
Epoch 9/20
385/385 [==============================] - 18s 47ms/step - loss: 0.2934 - accuracy: 0.9129 - val_loss: 0.5686 - val_accuracy: 0.8553
Epoch 10/20
385/385 [==============================] - 17s 45ms/step - loss: 0.2747 - accuracy: 0.9248 - val_loss: 0.5362 - val_accuracy: 0.8463
Epoch 11/20
385/385 [==============================] - 17s 45ms/step - loss: 0.2586 - accuracy: 0.9306 - val_loss: 0.5557 - val_accuracy: 0.8348
Epoch 12/20
385/385 [==============================] - 17s 43ms/step - loss: 0.2379 - accuracy: 0.9422 - val_loss: 0.6109 - val_accuracy: 0.8446
```

Slika 14: Treniranje dvosmjerne LSTM mreže kroz epohe



Slika 15: Rezultati treniranja dvosmjerne LSTM mreže

U slučaju treniranja dvosmjernog LSTM modela imamo problem preprilagođenosti (94.22% točnosti na skupu za treniranje i 84.46% na validacijskom skupu te 83.68% na skupu za testiranje) što se može zaključiti i iz vizualnog prikaza sa slike 15.

4.2.4. Integracija GloVe vektora sa dvosmjernom GRU mrežom

Kako bi koristili GloVe vektore moramo ih preuzeti i raspakirati sa službene stranice (`https://nlp.stanford.edu/data/glove.6B.zip`). Nakon što raspakiramo preuzetu mapu imamo pristup tekstualnim datotekama koje sadržavaju vektorske prikaze riječi u raznim dimanzijama (50, 100, 200 i 300). Za ovaj primjer ćemo koristiti prikaze sa 100 dimenzija koji se nalaze u `glove.6B.100d.txt` datoteci.

U isječku koda 19 prvo definiramo putanju do željene datoteke s vektorima. Nakon toga kreiramo rječnik `embeddings_index` u koji ćemo pohraniti vektore riječi u ((riječ, vektor) parovi). Svaki red u datoteci odgovara jednoj riječi i njezinoj vektorskoj reprezentaciji. Otvaramo datoteku i dijelimo svaki red na riječ (`values[0]`) i njezin odgovarajući vektor (`values[1:]`). Zatim pretvaramo vektor u `numpy` niz s brojevima u `float` formatu i pohranjujemo ga u rječnik `embeddings_index` s riječju kao ključem.

```
glove_path = './glove.6B.100d.txt'

embeddings_index = {}
with open(glove_path, 'r', encoding='utf8') as file:
    for line in file:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector
```

Isječak koda 19: Učitavanje vekorskih prikaza u riječnik; prema [10]

```
vocab = encoder.get_vocabulary()
embedding_dim = 100
embedding_matrix = np.zeros((len(vocab), embedding_dim))

for i, word in enumerate(vocab):
    if word in embeddings_index:
        embedding_matrix[i] = embeddings_index[word]
    else:
        embedding_matrix[i] = np.random.normal(scale=0.6, size=(embedding_dim, ))

embedding_layer = tf.keras.layers.Embedding(
        input_dim=len(vocab),
        output_dim=embedding_dim,
        weights=[embedding_matrix],
        trainable=False
)
```

Isječak koda 20: Kreiranje reprezentativne matrice i `embedding` sloja; prema [10]

U isječku 20 inicijaliziramo vocab na listu riječi iz našeg enkodera, te postavljamo `embedding_dim` na 100. `embedding_matrix` je reprezentativna matrica koja će sadržavati sve vektorske prikaze iz našeg vokabulara (inicijaliziramo je kao nul-matricu $M \in \mathbb{R}^{m \times n}$, gdje je m duljina vokabulara, a n veličina (dimanzija) prikaza).

Zatim sa funkcijom `enumerate` *enumeriramo* (indeksiramo) elemente liste `vocab` i za svaku riječ u vokabularu provjeravamo postoji li njen prikaz u ranije kreiranom `embeding_index`-u, odnosno postoji li GloVe prikaz dane riječi. Ako prikaz postoji stavljamo ga na i-tu poziciju (redak) u reprezentativnu matricu. U suprotnom, u red matrice stavljamo nasumično generirani vektor iz normalne distribucije sa standardnom devijacijom koja iznosi 0.6 (podaci će biti *skupljeniji* oko prosječne vrijednosti).

Na kraju kreiramo `embedding` sloj koji će u našem modelu kao rezultat vraćati vektorske prikaze riječi u sekvenci. Za parametar `weights` šaljemo kreiranu matricu te naglašavamo kako ne želimo podešavati ove težine tijekom procesa treniranja (`trainable=False`).

Kako bi mogli spremiti najbolje težine modela za kasniju upotrebu kreiramo `ModelCheckpoint` objekt i postavljamo ga tako da prati `val_loss`, te da spremi samo težine najboljeg modela na mjesto definirano s `checkpoint_path`.

```
checkpoint_path = "models/gru_glove.ckpt"

checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                monitor="val_loss", mode="min",
                                                save_best_only=True,
                                                save_weights_only=True,
                                                verbose=1)
```

Isječak koda 21: `checkpoint` za spremanje modela

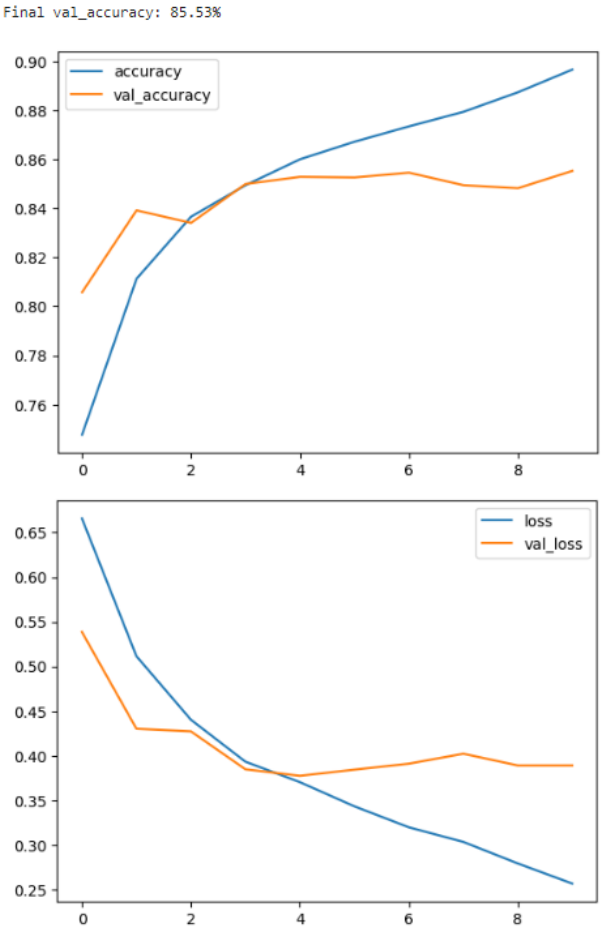
Nakon toga možemo definirati i trenirati GRU model kao i do sada, s tim da ćemo ovaj put koristiti ranije kreirani `embedding` sloj s GloVe vektorima.

```
gru_model_glove1 = tf.keras.models.Sequential([
        encoder,
        embedding_layer,
        Bidirectional(GRU(64)),
        Dense(32, activation='relu'),
        Dropout(0.4),
        Dense(3, activation='softmax')
])

early_stopping = tf.keras.callbacks.EarlyStopping(patience=5, monitor='val_loss')

gru_model_glove1.compile(optimizer=Adam(learning_rate=1e-3),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = gru_model_glove1.fit(train_ds, epochs=25, validation_data=val_ds,
↪  callbacks=[early_stopping, checkpoint])
```

Isječak koda 22: GRU model s GloVe vektorima

```
Epoch 1/25
384/385 [============================>.] - ETA: 0s - loss: 0.6659 - accuracy: 0.7476
Epoch 1: val_loss improved from inf to 0.53862, saving model to models/gru_glove.ckpt
385/385 [==============================] - 18s 35ms/step - loss: 0.6658 - accuracy: 0.7477 - val_loss: 0.5386 - val_accuracy: 0.8058
Epoch 2/25
385/385 [==============================] - ETA: 0s - loss: 0.5115 - accuracy: 0.8113
Epoch 2: val_loss improved from 0.53862 to 0.43024, saving model to models/gru_glove.ckpt
385/385 [==============================] - 12s 32ms/step - loss: 0.5115 - accuracy: 0.8113 - val_loss: 0.4302 - val_accuracy: 0.8392
Epoch 3/25
383/385 [============================>.] - ETA: 0s - loss: 0.4405 - accuracy: 0.8366
Epoch 3: val_loss improved from 0.43024 to 0.42728, saving model to models/gru_glove.ckpt
385/385 [==============================] - 12s 31ms/step - loss: 0.4403 - accuracy: 0.8366 - val_loss: 0.4273 - val_accuracy: 0.8341
Epoch 4/25
384/385 [============================>.] - ETA: 0s - loss: 0.3933 - accuracy: 0.8494
Epoch 4: val_loss improved from 0.42728 to 0.38481, saving model to models/gru_glove.ckpt
385/385 [==============================] - 12s 32ms/step - loss: 0.3933 - accuracy: 0.8494 - val_loss: 0.3848 - val_accuracy: 0.8499
Epoch 5/25
383/385 [============================>.] - ETA: 0s - loss: 0.3705 - accuracy: 0.8602
Epoch 5: val_loss improved from 0.38481 to 0.37750, saving model to models/gru_glove.ckpt
385/385 [==============================] - 12s 32ms/step - loss: 0.3704 - accuracy: 0.8600 - val_loss: 0.3775 - val_accuracy: 0.8529
Epoch 6/25
385/385 [==============================] - ETA: 0s - loss: 0.3433 - accuracy: 0.8672
Epoch 6: val_loss did not improve from 0.37750
385/385 [==============================] - 12s 32ms/step - loss: 0.3433 - accuracy: 0.8672 - val_loss: 0.3844 - val_accuracy: 0.8526
Epoch 7/25
383/385 [============================>.] - ETA: 0s - loss: 0.3202 - accuracy: 0.8732
Epoch 7: val_loss did not improve from 0.37750
385/385 [==============================] - 12s 32ms/step - loss: 0.3198 - accuracy: 0.8734 - val_loss: 0.3911 - val_accuracy: 0.8546
Epoch 8/25
383/385 [============================>.] - ETA: 0s - loss: 0.3032 - accuracy: 0.8794
Epoch 8: val_loss did not improve from 0.37750
385/385 [==============================] - 12s 32ms/step - loss: 0.3035 - accuracy: 0.8794 - val_loss: 0.4023 - val_accuracy: 0.8494
Epoch 9/25
384/385 [============================>.] - ETA: 0s - loss: 0.2792 - accuracy: 0.8874
Epoch 9: val_loss did not improve from 0.37750
385/385 [==============================] - 12s 32ms/step - loss: 0.2793 - accuracy: 0.8873 - val_loss: 0.3891 - val_accuracy: 0.8482
Epoch 10/25
385/385 [==============================] - ETA: 0s - loss: 0.2568 - accuracy: 0.8966
Epoch 10: val_loss did not improve from 0.37750
385/385 [==============================] - 13s 33ms/step - loss: 0.2568 - accuracy: 0.8966 - val_loss: 0.3891 - val_accuracy: 0.8553
```

Slika 16: Treniranje dvosmjerne GRU mreže sa GloVe vektorima kroz epohe



Slika 17: Rezultati treniranja dvosmjerne GRU mreže s GloVe vektorima

Na slikama 16 i 17 vidimo da ovaj model postiže dobre rezultate uz mali problem preprilagođensti. Na skupu za testiranje ovaj model je postigao bolji performans (86.12%) nego na validacijskom setu u zadnjij epohi (85.53%). Na slici 16 vidimo i kako se kroz proces treniranja spremaju najbolji modeli u danu datoteku.

4.2.5. Evaluacija GRU modela sa GloVe vektorima

Za evaluaciju modela možemo koristiti razne tehnike. Ovdje ćemo evaluirati naš model uz pomoć matrice zabune (engl. *confusion matrix*) i funkcije `classification_report`.

```
1 from sklearn.metrics import confusion_matrix, classification_report
```

Isječak koda 23: Uključivanje potrebnih modula

Nakon uvoza modula moramo dobiti klasifikacije modela na skupu za testiranje. To ćemo uraditi upotrebom `predict` metode. Rezultati će biti spremljeni u varijablu `y_pred` u obliku liste vektora vjerojatnosti. Da bi dobili rezultat u obliku cijelog broja (0, 1 ili 2) koji predstavlja ranije definirane klase (sentiment) moramo za svaki od tih vektora naći indeks na kojem mu se nalazi najveća vrijednost (`np.argmax`).

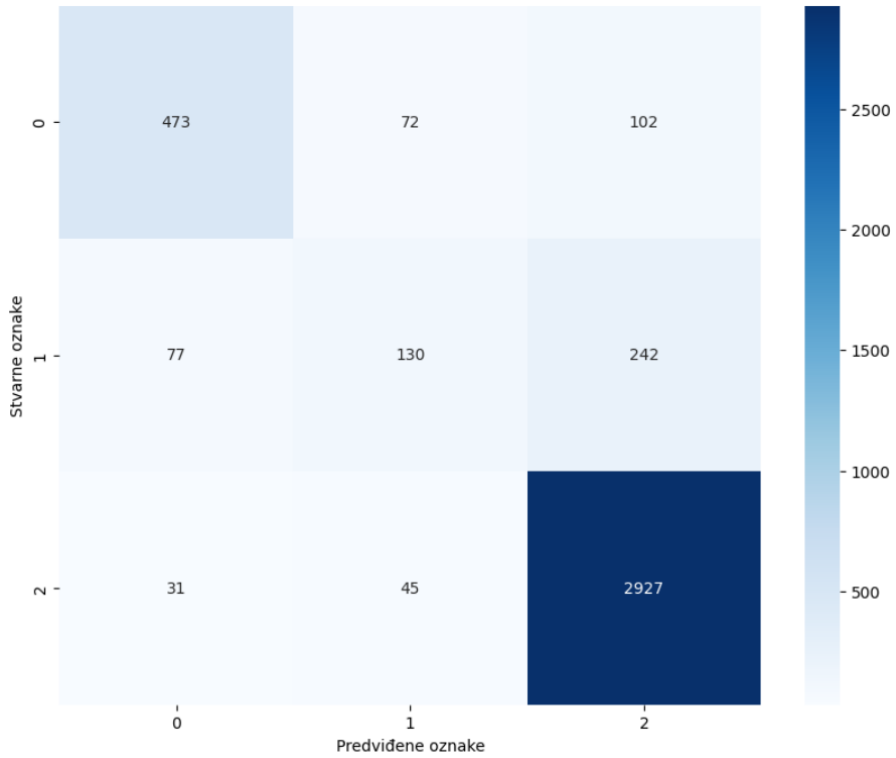
```
y_pred = gru_model_glove1.predict(X_test)
y_pred = tf.argmax(y_pred, axis=1).numpy()
```

Isječak koda 24: Pretvaranje klasifikacija u cijele brojeve

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predviđene oznake')
plt.ylabel('Stvarne oznake')
plt.title('Matrica zabune')
plt.show()
```

Isječak koda 25: Kreiranje i vizualizacija matrice zabune

Funkciji `conf_matrix` u isječku koda 25 šaljemo stvarne vrijednosti i klasifikacije našeg modela te uz pomoć `seaborn` modula kreiramo vizualizaciju dobivene matrice zabune.



Slika 18: Matrica zabune

Na matrici zabune vidljivo je da model najbolje prepoznaje pozitivne komentare (klasa 2), s 2927 točnih predviđanja. Manje je uspješan u razlikovanju negativnih (klasa 0) i neutralnih (klasa 1) recenzija, s obzirom na brojke kao što su 77 negativnih komentara predviđenih kao neutralne i 72 pozitivne recenzije klasificirane kao negativne.

Relativno visok broj lažno pozitivnih (engl. *false positive*, kada je komentar pogrešno klasificiran kao pozitivan) za negativne i neutralne klase (102 i 242) pokazuje da model ima tendenciju pogrešno klasificirati komentare kao pozitivne, što se može pripisati neravnoteži u distribuciji klasa. Ukupno gledano, model je relativno točan, ali pokazuje pristranost prema prepoznavanju pozitivnih komentara, što je vjerojatno zbog neravnoteže u skupu podataka.

Ovaj model možemo još evaluirati koristeći metrike koje se računaju pomoću matrice zabune (preciznost (engl. *precision*) i odaziv (engl. *recall*)). Ove mjere daju bolji uvid u performans modela jer je skup podataka, odnosno distribucija ocjena (slika 3) neravnomjerna (lijevo iskrivljena). Funkcija `classification_report` nam daje pregled ovih ključnih metrika.

	precision	recall	f1-score	support
0	0.81	0.73	0.77	647.00
1	0.53	0.29	0.37	449.00
2	0.89	0.97	0.93	3003.00
accuracy	0.86	0.86	0.86	0.86
macro avg	0.75	0.67	0.69	4099.00
weighted avg	0.84	0.86	0.85	4099.00

Slika 19: *Classification report*

Za klasu 0 (negativno) imamo visoku preciznost (0.81) što znači da kada model predvidi komentar kao negativan, uglavnom je klasifikacija točna. Međutim, odaziv od 0.73 ukazuje da model propušta neke negativne komentare (nisu klasificirani kao negativni).

Klasa 1 (neutralno) ima nisku preciznost (0.53) i odaziv (0.29), što znači da model ne prepoznaje dobro neutralne komentare, te često pogrešno klasificira komentare koji pripadaju ovoj kategoriji.

Klasa 2 (pozitivno) ima najveću preciznost (0.89) i odaziv (0.97) što ukazuje na to da model vrlo dobro prepoznaje pozitivne komentare i rijetko ih krivo klasificira.

Ukupna točnost modela je 0.86, što znači da model ispravno klasificira recenzije u 86% slučajeva.

4.3. Prikaz rada aplikacije

Naš model je sada spreman za klasifikaciju novih recenzija. Kako bi koristili težine koje smo naučili u prikazanom procesu treniranja (slika 16) trebamo definirati i kompajlirati model s istom strukturom kao onaj u isječku koda 22 i u njega učitati spremljene težine (metoda `load_weights`).

```
gru_model_glove = tf.keras.models.Sequential([
        encoder,
        embedding_layer,
        Bidirectional(GRU(64)),
        Dense(32, activation='relu'),
        Dropout(0.4),
        Dense(3, activation='softmax')
])

gru_model_glove.compile(optimizer=Adam(learning_rate=1e-3),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

gru_model_glove.load_weights(checkpoint_path)
```

Isječak koda 26: Definiranje, kompilacija i učitavanje spremljenih težina u novi model

Definirat ćemo i funkciju koja će kao argumente primati recenziju i model za klasifikaciju. Ova funkcija će procesirati danu recenziju koristeći funkciju definiranu u isječku koda 8, pretvoriti procesiranu recenziju u TensorFlow skup podataka (`tf.data.Dataset`), klasificirati recenziju i vratiti rezultat u čitljivom obliku.

```
def classify_comment(comment, model):
    labels = ['Negative', 'Neutral', 'Positive']

    # Pretprocesiranje recenzije
    preprocessed_comment = custom_preprocessing_sw_removal(comment)

    # Pretvaranje u TensorFlow Dataset
    ds = tf.data.Dataset.from_tensor_slices([preprocessed_comment])
    ds = ds.batch(1)

    # Klasifikacija
    prediction = model.predict(ds)

    return labels[np.argmax(prediction)]
```

Isječak koda 27: Funkcija za klasifikaciju recenzija

Ovu funkciju možemo koristiti na sljedeći način:

```
comment = "I was very disappointed with my stay. The room was dirty and the service
↪  was unacceptably slow."
prediction = classify_comment(comment, gru_model_glove)
print(f"Predicted Sentiment: {prediction}\n")
```

Isječak koda 28: Primjer korištenja funkcije za klasifikaciju recenzije

Kao izlaz prethodnog isječka dobijemo:

```
1/1 [==============================] - 1s 611ms/step
Predicted Sentiment: Negative
```

Slika 20: Klasifikacija recenzije

Vidimo da funkcija za klasifikaciju ispravno radi, te se sada možemo osvrnuti na gotovo rješenje i predložiti neka poboljšanja.

4.4. Kritički osvrt

Na temelju evaluacije, jasno je da sustav pokazuje pristranost prema pozitivnim komentarima. Osim toga, vidjeli smo kako model pokazuje slabije rezultate u klasifikaciji neutralnih komentara, što može biti znak da je razlikovanje između neutralnog i ekstremnog sentimenta (pozitivnog ili negativnog) izazovno za trenutni model.

Jedan od mogućih pristupa poboljšanju ovakvog sustava je isprobavanje drugih pristupa strojnog učenja. Naive Bayes klasifikator bi se mogao isprobati kao alternativa metodama dubokog učenja kao što su GRU, LSTM i rekurentna neuronska mreža.

Kako se neutralni komentari pokazuju kao teško razlučivi, mogli bismo isprobati i preoblikovanje problema iz višeklasne klasifikacije u binarnu (pozitivno/negativno) što bi moglo smanjiti složenost modela i povećati njegovu preciznost.

Postojeće rješenje bi mogli probati poboljšati i podešavanjem parametara koji utječu na učenje (engl. *hyperparameter tuning*. To uključuje eksperimentiranje s brojem slojeva u mreži, brojem jedinica u skrivenim slojevima, stopom učenja i drugim relevantnim parametrima.

Još jedan pristup bi bio upotreba naprednijih NLP modela, kao što je BERT (*Bidirectional Encoder Representations from Transformers*), koji su pokazali izvanredne rezultate na različitim NLP zadacima.

5. Zaključak

U okviru ovog seminarskog rada, istražili smo i implementirali sekvencijalne modele za klasifikaciju sentimenta u ugostiteljskoj industriji, koristeći Python i TensorFlow.

Implementacija modela u TensorFlow-u omogućila nam je korištenje sofisticiranih alata koji su značajno olakšali proces modeliranja i eksperimentiranja što je doprinijelo bržem iteracijskom ciklusu u izgradnji ovog sustava.

Primjena ovog NLP sustava u ugostiteljskoj industriji može se odnositi na analizu recenzija restorana, hotela i barova, pružajući uvide u mišljenja i preferencije klijenata. Ovi uvidi su ključni za poboljšanje kvalitete usluge, razvoj menija i unapređenje općeg iskustva klijenata. Također, mogu poslužiti za praćenje i reagiranje na trendove u industriji, te za identifikaciju i rješavanje specifičnih problema koji mogu negativno utjecati na reputaciju poslovanja.

Popis literature

[1] J. Chung, C. Gulcehre, K. Cho i Y. Bengio, „Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[2] A. Karpathy. „The Unreasonable Effectiveness of Recurrent Neural Networks." (2015.), adresa: https://karpathy.github.io/2015/05/21/rnn-effectiveness/ (pogledano 12. 1. 2024.).

[3] A. Ng. „Bidirectional RNN," Coursera. (2021.), adresa: https://www.coursera.org/learn/nlp-sequence-models/lecture/fyXnn/bidirectional-rnn (pogledano 9. 1. 2024.).

[4] C. Olah. „Understanding LSTM Networks." (2015.), adresa: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (pogledano 10. 1. 2024.).

[5] S. Kostadinov. „Understanding GRU Networks," Medium. (2017.), adresa: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be (pogledano 10. 1. 2024.).

[6] A. Ng. „Using Word Embeddings," Coursera. (2021.), adresa: https://www.coursera.org/learn/nlp-sequence-models/lecture/qHMK5/using-word-embeddings (pogledano 9. 1. 2024.).

[7] A. Ng. „Properties of Word Embeddings," Coursera. (2021.), adresa: https://www.coursera.org/learn/nlp-sequence-models/lecture/S2mat/properties-of-word-embeddings (pogledano 9. 1. 2024.).

[8] J. Pennington, R. Socher i C. D. Manning, „Glove: Global vectors for word representation," *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014., str. 1532–1543.

[9] S. Cloud. „Stemming in Natural Language Processing." (), adresa: https://saturncloud.io/glossary/stemming/#:~:text=Stemming%20is%20a%20text%20preprocessing,classification%2C%20and%20other%20NLP%20tasks. (pogledano 12. 1. 2024.).

[10] SkillC, *Sentiment Analysis with LSTM | Deep Learning with Keras | Neural Networks | Project8*, 2022.

Popis slika

Popis tablica

Popis isječaka koda