UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Nenad Mišić, R2-19/2020
Dušan Milunović, R2-20/2020
Branislav Anđelić, R2-21/2020

Pretraživanje zakona Republike Srbije

Seminarski rad
- Master akademske studije -

Novi Sad, 2020.

SADRŽAJ

1. UVOD

Zadatak ovog rada biće napraviti program za pretragu zakona Republike Srbije. Poznavanje zakona Republike Srbije je u okviru opšte populacije na veoma niskom nivou. Ovakav sistem pretrage bi olakšao pristup zakonima ljudima koji nisu u pravnoj struci, skraćivanjem potrebnog vremena za pronalazak odgovarajućeg (skupa) zakona za neki realan problem. Pored toga, cilj autora je uporediti različite metode pretrage u ovom polju.

Sistemi za pretragu dokumenata se mogu praviti na različite načine. Neki od njih bili bi pretraga po čistom tekstu, pretraga po meta-podacima ili pretraga uz pomoć mašinskog učenja. U ovom radu fokus će biti na pretrazi dokumenata uz pomoć mašinskog učenja, odnosno na poređenju različitih metoda na ovom zadatku.

Metode mašinskog učenja koje su rađene su: *TF-IDF*, *word2vec*, *doc2vec* i *XLM-Roberta transformer*. Ono što je zajedničko za sve njih jeste da ne rade sa čistim tekstom, već pretvaraju tekst u vektore. Nakon predstavljanja zakona pomoću vektora, pretraga postaje trivijalna. Potrebno je samo predstaviti korisnikov upit kao vektor, i pronaći onaj zakon čiji je vektor najbliži upitu. Svaka od metoda rešava problem predstavljanja teksta vektorom na drugačiji način.

Osim njih urađena je pretraga uz pomoć *Elasticsearch*-a. *Elasticsearch* je alat otvorenog koda za pretragu sadržaja, bilo tekstualnog, numeričkog, strukturiranog ili nestrukturiranog. Pretraga uz pomoć dokazano kvalitetnog alata je bila dobra početna tačka, kao i dobra referentna tačka za poređenje performansi ostalih rešenja.

Skup podataka se sastoji od 1130 zakona. Evaluacija je rađena nad skupom od 11 upita sa ljudski evaluiranim očekivanim rezultatima, korišćenjem dve mere - odziv i *Spearman-ova* mera preciznosti.

U narednom poglavlju biće date teorijske osnove svakog od korišćenih algoritama. U trećem poglavlju biće opisana implementacija rešenja. U četvrtom poglavlju biće dat pregled metoda evaluacije i rezultata evaluacije svih rešenja. Poglavlje pet će biti zaključak rada.

2. TEORIJSKI POJMOVI I DEFINICIJE

U ovom poglavlju biće objašnjeni osnovni teorijski pojmovi i definicije potrebni za razumevanje načina rada programa. Za početak biće objašnjeno kako radi *Elasticsearch* (poglavlje 2.1). Nakon toga biće objašnjen *TF-IDF* (poglavlje 2.2), jer je on najjednostavniji način za predstavljanje teksta vektorom koji je korišćen. Zatim će biti opisani *Word2vec* (poglavlje 2.3), *Doc2vec* (poglavlje 2.4) i *XLM-Roberta transformer* (poglavlje 2.5), koji predstavlja *state-of-the-art* model u polju obrade prirodnog jezika.

2.1 Elasticsearch

Apache Lucene [10] je softver otvorenog koda namenjen za pretragu celokupnog teksta pomoću tehnike indeksiranja. Pruža mogućnost pretrage kako strukturiranih tako i nestrukturiranih podataka. *Lucene* je poslednjih godina najpopularnija besplatna *Java* biblioteka za pretragu teksta.

Elasticsearch [11] je distribuirani softverski alat otvorenog koda zasnovan na *Apache Lucene*. Pruža mogućnost pretrage različitih tipova podataka, kao što su tekstualni, numerički, geoprostorni, stukturirani i nestrukturirani. Za razliku od *Lucene*, koji pruža *Java API,Elasticsearch* pruža *RESTful API* za rukovanje podacima u *JSON* formatu, pa je korišćenje van *Java* okruženja znatno jednostavnije.

Elasticsearch funkcioniše kao skladište dokumenata (kao dokument-orijentisane *NoSQL* baze podataka). U poređenju sa tradicionalnim načinom čuvanja podataka u tabelarnoj formi, ovaj pristup pruža veću fleksibilnost u skaliranju i distibuciji. Za pretragu se koristi struktura inverznog indeksa koja, čuvanjem liste dokumenata za svaku jedinstvenu reč, pruža efikasan način pretrage velikog broja dokumenata. Takođe, *Elasticsearch* pruža podršku za nestrukturirane dokumente, bez specificirane šeme, što je u slučaju našeg projekta neophodno svojstvo.

2.2 TF-IDF

TF-IDF [8] je akronim za *Term frequency – Inverse document frequency*. On predstavlja unapređenje algoritma *bag-of-words*.

Bag-of-words je jedan od najjednostavnijih algoritama za predstavljanje dokumenata preko vektora. Prvi korak je da se prođe kroz sve dokumente i da se sakupi skup jedinstvenih reči u vokabular. Nakon toga, opet se prolazi kroz svaki dokument i oni se predstavljaju kao vektor dužine

vokabulara. U tom vektoru se na svakom indeksu nalazi broj pojavljivanja određene reči u tekstu. Mane ovog pristupa su što se ne uzima u obzir učestalost reči (neke reči se u jeziku pojavljuju vrlo često dok su druge retke), kao ni dužina dokumenta (duži dokumenti će imati vektore sa višim vrednostima) i što su vektori dokumenata većinski ispunjeni nulama.

TF-IDF popravlja prve dve mane *bag-of-words* algoritma. Prvi deo algoritma je isti, s tim da se osim brojanja reči vokabulara, za svaku reč beleži u kom broju dokumenata se ona javlja. To je *document frequency* reči, koji je velik za učestale reči, a mali za retke. Za računanje vrednosti vektora teksta više se ne koristi samo broj pojavljivanja svake reči. Umesto toga, koristi se broj pojavljivanja svake reči u odnosu na dužinu teksta (to predstavlja *term frequency*) koji se zatim množi sa *inverse document frequency* za svaku reč. Na ovaj način, dugačkim dokumentima se smanjuju vrednosti pojedinačnih reči, i smanjuje se vrednost reči koje se pojavljuju u velikom broju dokumenata.

2.3 Word2vec

Word2vec [1][2] je jedan od najznačajnijih algoritama u razvoju obrade prirodnog jezika. On se zasniva na principu da se reč može opisati kroz svoj kontekst. Jedna bitna odlika ovog algoritma jeste da izlazni vektori više ne sadrže po jednu vrednost za svaku reč iz vokabulara, već je svaka reč predstavljena sa vektorom, a dokument se vektorizuje tako što se agregiraju vektori svih njegovih reči. Ovo znači da dokumenti mogu da se predstave sa vektorima daleko manje dimenzionalnosti nego kod *TF-IDF* (uobičajene dimenzije vektora su između sto i trista kod *word2vec,* a kod *TF-IDF* od nekoliko desetina hiljada pa sve do miliona).

Sam algoritam je neuronska mreža koja se može trenirati na dva načina. Prvi način je da se kao ulaz primi kontekst reči (nekoliko okolnjih reči), a kao izlaz mreža pokušava da pogodi koja je reč u pitanju, a drugi je da se kao ulaz mreže da reč, a kao izlaz mreža pokušava da pogodi koje su reči oko nje. Na taj način, neuronska mreža uči vektorske predstave reči. U oba slučaja, model može da se trenira nad neoznačenim tekstom i s obzirom na ogromne količine dostupnih podataka u današnje vreme, mreža vrlo dobro uspeva da nauči vektore za reči.

Kao što je napomenuto, kako bi se dobio vektor za dokument, potrebno je agregirati vektore reči. Jedan od načina da se to uradi jeste da se uzme prosečna vrednost vektora svih reči dokumenta.

2.4 Doc2vec

Doc2vec [2] je nastao kao naslednik *word2vec*-a. Poenta *doc2vec*-a je bila da se nivo posmatranja dokumenta podigne sa jedne reči na čitav dokument. To se postiže tako što se kao ulaz mreže dodaje još jedan token osim reči, a to je identifikator dokumenta. Tokom treniranja, cilj mreže je da iz identifikatora dokumenta i podskupa reči koje on sadrži pogodi jednu od reči iz dokumenta koja se ne nalazi na ulazu mreže. Na taj način, osim vektora reči, uči se i vektor samih dokumenata.

Kada je reč o upotrebi mreže za određivanje vektora nepoznatih dokumenata, dolazi do problema što za njih ne postoji unapred određen identifikator. Rešenje je da se mreža pokrene sa nasumično odabranim identifikatorom na ulazu, i sa podskupom reči koje on sadrži, i da se proveri da li se kao izlaz pojavljuje neka od njegovih preostalih reči. Ukoliko se ne nalazi, koristeći *backpropagation* namešta se identifikator dokumenta sve dok se ne dobije željeni izlaz. Kada se to desi, smatra se da dokument ima odgovarajući identifikator i da je mreža spremna da odredi njegov vektor.

Iz principa rada *doc2vec*-a proizilaze dva problema. Prvi je taj što se početni identifikator dokumenta bira nasumično, te se za isti dokument uvek dobija nešto drugačiji vektor. Drugi problem je što se *backpropagation* ponavlja veliki broj puta, pa mreža radi dosta sporo. To je predstavljalo veliko ograničenje *doc2vec* pristupa za rešavanje zadatka ovog rada, jer je potrebno odrediti vektore svih zakona.

2.5 XLM-Roberta transformer

XLM-Roberta [3] je *state-of-the-art* transformer model. *Transformer* modeli su poslednjih nekoliko godina najbolji modeli u oblasti obrade prirodnog jezika, a prvi put su predstavljeni u radu o *Google*-ovom *BERT* [4] modelu. Nakon *BERT*-a, pojavio se *RoBERTa* [5] model koji je pokazao da se mogu postići još bolji rezultati ukoliko se trening izvršava na većem skupu podataka. *Facebook AI* je prateći istu logiku razvio *XLM-Roberta* model. Glavna razlika je u tome što je njihov model treniran nad skupom podataka koji se sastoji od tekstova na preko 100 različitih jezika. Taj skup podataka se sastoji od 2.5 TB teksta, a sam model dolazi u dve varijante, gde jedna ima 270 miliona parametara, a druga 550 miliona parametara. Razlog zašto je *XLM-Roberta* relevantan za naš rad je što je između ostalih, treniran i na srpskom jeziku na ćiriličnom pismu, a većina zakona je pisana ćirilicom.

Transformer modeli su zasnovani na *encoder-decoder* [6] arhitekturi. Ona radi uz pomoć dve *LSTM* [7] rekurentne mreže, gde prva (*encoder*) prođe kroz celu sekvencu (u slučaju ovog rada tekst) kako bi izvukla informacije o ulazu, a zatim radi generisanje izlaza drugom mrežom. Problem *encoder*-

decoder arhitekture nastaje kada su ulazne sekvence previše dugačke jer onda *encoder* mreža ne može da „upamti“ sve podatke. *Transformer* modeli rešavaju taj problem koristeći „pažnju“ (engl. *attention*) [4]. Ta modifikacija znači da *decoder* osim izlaza *encoder*-a kao ulaz koristi pažnju, koja označava koliko su određeni delovi ulazne sekvence bitni pri generisanju delova izlazne sekvence.

Ovi modeli se mogu dotrenirati (engl. *fine-tune*) za različite zadatke. U ovom radu iskorišćeni su za zadatak predstavljanja dokumenta kao vektora.

3. SPECIFIKACIJA I IMPLEMENTACIJA REŠENJA

Ovo poglavlje posvećeno je analizi i obradi skupa podataka (poglavlje 3.1), korišćenim alatima i bibliotekama (poglavlje 3.2), specifikaciji rešenja i arhitekturi sistema (poglavlje 3.3).

3.1 Skup podataka

Skup podataka sastoji se iz 1130 dokumenata zakona Republike Srbije. Svaki zakon predstavljen je html datotekom. Pretprocesiranje skupa podataka sastoji se iz nekoliko koraka:

- Zamenjivanje velikih slova u tekstu malim
- Uklanjanje html tagova
- Uklanjanje znakova interpunkicje
- Uklanjenje viška razmaka i novih redova
- Uklanjanje stop reči i numeričkih karaktera

Takođe, iz dokumenata su izvlačeni i naslovi zakona na osnovu njihovog položaja u html datoteci, koji predstavljaju metapodatak zakona.

Za *XLM-Roberta, word2vec* i *doc2vec* algoritme pretrage odrađena je dodatna transformacija svakog dokumenta u niz članova zakona, dok *Elasticsearch* i *TF-IDF* kao ulaz imaju čitav tekstualni sadržaj dokumenta.

3.2 Arhitektura rešenja

U ovoj sekciji navedena je arthitektura rešenja za svaki od metoda pretrage.

3.2.1 Elasticsearch

S obzirom da *Elasticsearch* pruža baš one funkcionalnosti koje su nama potrebne, ovaj alat smo iskoristili kao početnu i referentnu tačku u razvoju našeg rešenja. Dokument, u našem slučaju, predstavlja zakon kao neuređeni tekst. Bilo je potrebno uraditi i preprocesiranje, jer se u skupu podataka nalaze zakoni u html obliku, pa je bilo potrebno ukloniti html tagove. Pored teksta, kao poseban atribut čuva se i naziv zakona kao i putanja do html datoteke u kojoj je sadržan. Slanjem upita ka *Elasticsearch* serveru, vrši se pretraga svih dokumenata nad poljima koja sadrže tekst i naslov zakona. Kako je u pitanju obična pretraga teksta, sva podešavanja su ostavljena kao podrazumevana.

3.2.2 TF-IDF

Za implementaciju *TF-IDF* algoritma upotrebljena je *TfidfVectorizer* klasa iz *sci-kit learn* [16] *python* biblioteke. Ulaz u algoritam predstavlja niz dokumenata u tekstualnoj reprezentaciji, gde su obeležja algoritma pojedinačne reči. Za regularizaciju upotrebljena je *L2* norma. Kako bi se sprečila mogućnost deljenja nulom, algoritam je podešen da frekventnosti svake reči doda 1, kao da postoji dodatni dokument koji sadrži sve reči. Kako reči imaju značajan broj ponavljanja unutar istog dokumenta u skupu podataka, primenjeno je sublinearno skaliranje nad vrednostima učestalosti reči *tf*, koja *tf* transformiše u *1 + log(tf)*. Ovim se daje manja težina većem broju ponavljanja reči unutar dokumenta, što je pokazalo kao unapređenje performansi algoritma.

Nad upitom odrađena je ista transformacija kao i nad skupom podataka, nakon čega se rezultati dobijaju kao lista dokumenata rangiranih po kosinusnoj sličnosti *tf-idf* vektora upita sa dokumentom. Iz ove liste odabrano je deset dokumenata sa najvećom merom sličnosti kao konačan rezultat pretrage.

3.2.3 Word2vec

Treniranje *Word2vec* modela rađeno je upotrebom *Word2Vec* klase iz *gensim* [12] *python* biblioteke. Trening podaci predstavljaju listu članova svih zakona iz pretprocesiranog skupa podataka. Veličina vektora predstave reči podešena je na 300, gde za svaku reč model traži sličnosti u prozoru od pet susednih reči. Minimalni broj pojavljivanja reči u dokumentu je postavljen na 1, tako da model svakoj reči iz skupa podataka dodeljuje vektorsku predstavu.

Vektorske predstave reči sačuvane su u datoteku, kako bi bile učitane u *web* aplikacije, čime se izbegava ponovno treniranje modela prilikom podizanja aplikacije. Nad upitom odrađena je ista transformacija kao i nad skupom podataka, nakon čega *word2vec* model transformiše upit u vektorsku reprezentaciju. Rezultati pretrage se dobijaju kao lista dokumenata rangiranih po prosečnoj kosinusnoj sličnosti vektorske reprezentacije upita sa vektorskom reprezentacijom tri najsličnija člana unutar dokumenta. Iz ove liste odabrano je deset dokumenata sa najvećom merom sličnosti kao konačan rezultat pretrage.

3.2.4 Doc2vec

Treniranje *Doc2vec* modela rađeno je upotrebom *Doc2vec* klase iz *gensim* [12] *python* biblioteke. Trening podaci predstavljaju listu članova svih zakona iz pretprocesiranog skupa podataka, gde je u pogledu *doc2vec* algoritma svaki član jedan dokument. Veličina vektora predstave dokumenta podešena je na 300. Minimalni broj pojavljivanja reči u dokumentu je postavljen na 1, tako da model posmatra svaku reč iz skupa podataka. Po zahtevu algoritma, svakom članu dodeljen je identifikacioni broj, upotrebom *TaggedDocument* klase iz *gensim* biblioteke.

Vektorske predstave članova sačuvane su u datoteku, kako bi bile učitane u web aplikacije, čime se izbegava ponovno treniranje modela prilikom podizanja aplikacije. Upit je posmatran kao dokument, koji se vektorizuje pomoću modela nakon izvršenih transformacije pretprocesiranja upita. Rezultati pretrage se dobijaju kao lista dokumenata rangiranih po prosečnoj kosinusnoj sličnosti vektorske reprezentacije upita sa vektorskom reprezentacijom tri najsličnija člana unutar dokumenta. Iz ove liste odabrano je deset dokumenata sa najvećom merom sličnosti kao konačan rezultat pretrage.

3.2.5 Xlm-roberta

Pretrenirani model *Xlm-roberta* učitan je pomoću *Sentence transformers* [19] bilbioteke za *python* programski jezik, uz pokretanje na grafičkom procesoru ukoliko *hardware* uređaja to dozvoljava. Ulaz u model je lista članova za svaki zakon, a izlaz vektorska reprezentacija tih članova. Ove reprezentacije sačuvane su u datoteci kako bi se izblegao proces vektorizacije prilikom svakog pokretanja web aplikacije.

Pretraga po upitu vrši se vektorizacijom pretprocesiranog teksta upita korišćenjem istog modela kao i za vektorizaciju skupa podataka. Vektorizovani upit poređen je sa vektorskom reprezentacijom članova merom kosinusne sličnosti. Rezultat pretrage predstavljaju zakoni sa najvećom prosečnom sličnosti tri svoja člana sa najvećom vrednosti mere sličnosti.

3.3 Korišćeni alati i biblioteke

Sistem je implemetiran sa ciljem da demonstrira rešenje problema tekstualne pretrage zakona, a ne da sam po sebi bude gotov proizvod spreman za korišćenje. Shodno tome, korišćeni alati, biblioteke i radni okviri su svedeni na minimum kako bi se problem koji se rešava lokalizovao.

Ceo sistem se može podeliti u tri odvojena dela:

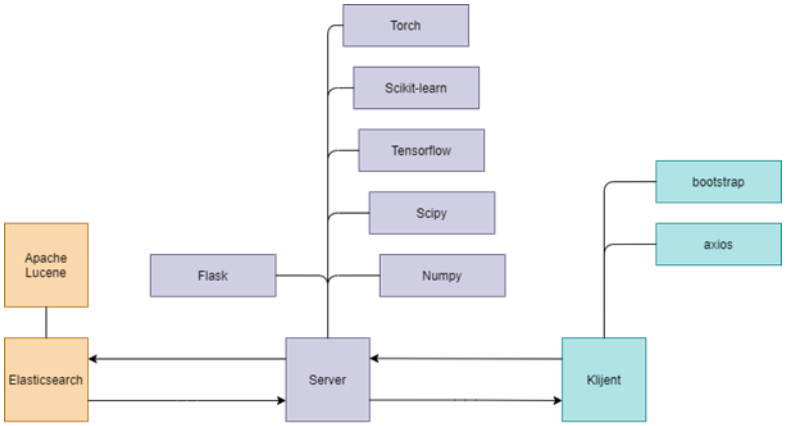
- Klijentska aplikacija
- Serverska aplikacija
- *Elasticsearch* servis

Ukoliko govorimo o serverskom delu, tu je reč o *Python* aplikaciji koja se preko *Pypi* alata za upravljanje zavisnostima oslanja na *Flask* [18] biblioteku. To je biblioteka koja omogućuje jednostavno i brzo kreiranje web servera sa *RESTful* komunikacijom. Pored toga, korišćen je i čitav niz biblioteka koje su standard u oblasti veštačke inteligencije, kao što su *scipy* [14], *numpy* [15], *scikit-learn* [16], *gensim* [12], *torch* [13] i *tensorflow* [17].

Što se klijentskog dela tiče, to je jednostavna aplikacija namenjena za pokretanje unutar internet pretraživača, a njoj se pristupa kao statičkom resursu, preko serverske aplikacije, koja je odgovorna za njeno serviranje. Da bi se smanjila kompleksnost sistema, pri implementaciji je izbegnuto korišćenje dodatnih radnih okvira, već se rešenje zasniva na tzv. *Vanilla* verziji *Javascript* jezika. Za potrebe *HTTP* komunikacije sa serverom iskorišćena je *axios* biblioteka. Naposletku, stilizovanje klijentske aplikacije postignuto je uz pomoć *Bootstrap* alata.

Elasticsearch servis je podignut kao lokalno, nedistribuirano skladište čijim se funkcionalnostima pristupa kroz *Java* server koji je obezbeđen kao gotov alat od strane *Elastic*-a, kompanije koja stoji iza Elasticsearch alata. Komunikacija serverskog sloja sa ovim slojem takođe je realizovana preko *HTTP* protokola.

Korišćeni alati pri implementaciji sistema ilustrovani su na slici 3.1



Slika 3.1 Korišćeni alati pri implementaciji sistema

4. EVALUACIJA

Evaluacija pretraživača se zasniva na dve osnovne mere – preciznost i odziv. Preciznost je proporcija dobavljenih dokumenata koji su relevantni, a odziv proporcija broja dobavljenih relevantnih dokumenata. Relevantnost dokumenata zasnovana je na ljudskoj evaluaciji. [9] U ovom radu, relevatni dokumenti su odabrani kao podskup skupa dobavljenih dokumenta svih modela pretraživača, i rangirani su po nivou relevantnosti. Ovaj postupak je odrađen za 11 upita.

Korišćene su dve mere za evaluciju pretraživača – odziv *r* i preciznost *p*. Odziv predstavlja odnos broja dobavljenih relevantnih dokumenta i broja ukupnih relevatnih dokumenta. Za jedan upit, odziv se računa izrazom:

$$r = \frac{tp}{n}$$

gde je *tp* broj dobavljenih relevatnih dokumenata, a *n* broj ukupnih relevantnih dokumenata.

Odziv pretraživača za svaki upit nalazi se u Tabeli 1.

upit	XLM-r	elastic	w2v	d2v	tfidf
1	0.2	0.5	0.3	0	0.3
2	1	1	1	0	1
3	0.7	0	0.6	0.1	0
4	1	1	0.67	0	0.67
5	0.67	0.67	0	0	0.67
6	0.7	0.6	0.1	0	0.5
7	0.62	0.54	0.54	0	0.54
8	0.38	0.75	0.12	0.12	0.38
9	0.4	0.2	0.1	0	0.3
10	1	1	0.8	0.2	0.8
11	1	0.33	0.67	0	0.33

Tabela 1. Tabelarni prikaz odziva pretraživača

Odziv jednog pretraživača dobija se primenjivanjem navedenog izraza za svaki upit, odnosno formulom:

$$R = \frac{\sum_i^t tp_i}{\sum_i^t n_i}$$

gde je *t* broj upita.
Ukupan odziv svakog pretraživača nalazi se u Tabeli 2.

XLM-r	0.59
elastic	0.5
w2v	0.37
d2v	0.04
tfidf	0.41

Tabela 2. U

Kako je retultat pretrage svakog pretraživača rangirana lista rezultata, za određivanje preciznosti upotrebljena je *Spearman*-ova mera korelacije rankova, koja se za jedan upit računa po formuli:

$$S = 1 - \frac{6\sum_i^n d_i^2}{n(n^2 - 1)}$$

gde je *d* distanca, n broj rezultata u upitu. Distanca predstavlja razliku ranka rezultata pretrage od ranka rezultata ljudske evaluacije. Ukoliko se neki rezultat pretraživača ne nalazi u rezultatima ljudske evaluacije, distanca ta taj rezultat se postavlja na maksimalnu vrednost, odnosno $d = n$. Preciznost svakog pretraživača po upitima može se naći u Tabeli 3. Ideja ove mere performanse je da zaključi koliko dobro neki pretraživač rangira pronađene rezultate.

upit	XLM-r	elastic	w2v	d2v	tfidf
1	-0.65	-0.11	-0.33	-1.36	-0.65
2	1.36	1.42	1.42	1.36	1.42
3	0.87	-1.36	-0.38	-1.15	-1.36
4	2.18	2.24	2.18	2.45	2.35
5	2.35	2.4	2.45	2.45	2.13
6	0.71	0.44	-1.04	-1.36	-0.27
7	0.11	-0.71	-0.82	-5.18	-0.98
8	-1.36	-0.55	-2.18	-2.13	-1.53
9	-0.33	-0.87	-1.53	-1.36	-0.38
10	0.6	0.44	0.22	-0.55	0.33
11	2.45	2.4	2.18	2.45	2.4

Tabela 3. Preciznost svakog predtraživača po upitima

Generalno merilo preciznosti pretraživača dobijeno je kao prosečna vrednost preciznosti za svaki upit, i može se naći u Tabeli 4.

XLM-r	0.75
elastic	0.52
w2v	0.2
d2v	-0.4
tfidf	0.31

Tabela 4. Prosečne vrednosti preciznosti za svaki pretraživač

Iz rezultata evaluacije može se zaključiti da najbolje performnase ima pretraživač koji koristi *XLM-Roberta* neuronski mrežu, sa odzivom 0.59 i merom preciznosti od 0.75. Nakon njega sledi *Elasticsearch* sa uporedivim odzivom ali znatno manjom preciznosti. Ubedljivo najlošije rezultate pokazuje *doc2vec* implementacija pretraživača, sa odzivom od 0.04 i merom preciznosti od -0.4.

5. ZAKLJUČAK

U ovom radu predstavljen je sistem za pretragu zakona Republike Srbije i nekoliko reprezentativnih algoritama za pretragu celokupnog teksta, kao što su *Word2vec, XLM-Roberta, Elasticsearch* i drugi.

Motivacija za ovaj rad krije se u tome što je spektar zakona i pravnih dokumenata ogroman i, s obzirom da slično rešenje ne postoji kao javno dostupno, pronalaženje traženog zakonskog akta oduzima previše vremena i neretko zahteva profesionalnu pomoć pravnika ili advokata.

Sistem predložen u radu sastoji se od tri komponente, serverske aplikacije, klijentske aplikacije i *Elasticsearch* servisa. Sistem je, samim tim, lak za preztentaciju, na jednostavan način vizualizuje postignute rezultate i spreman je za upotrebu nad podskupom slučajeva korišćenja.

Istraživanje na ovu temu bi se moglo proširiti razmatranjem ideje o detaljnijoj ekstrakciji metapodataka iz dokumenata, kao i implementaciji detaljne pretrage po istim. Na taj način bi mogli da se zanemare zakoni koji trenutno nisu aktuelni ili pretraže samo zakoni doneti u zadatom vremenskom intervalu. Takođe, bilo bi korisno i prikupiti dodatne podatke kako bi se dalje proširio skup podataka, koji je trenutno relativno siromašan. Još jedna od ideja za dalje istraživanje bila bi podrška za zakone drugih država, jer pomenute metode ni na koji način nisu vezane za srpski jezik.

6. LITERATURA

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

[2] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546.*

[3] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... & Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116.*

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

[5] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692.*

[6] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

[7] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

[8] Mishra, A., & Vishwakarma, S. (2015, December). Analysis of tf-idf model and its variant for document retrieval. *In 2015 international conference on computational intelligence and communication networks (cicn) (pp. 772-776). IEEE.*

[9] Vaughan, L. (2004). New measurements for search engine evaluation proposed and tested. *Information Processing & Management, 40(4), 677-691.*

[10] Białecki, A., Muir, R., Ingersoll, G., & Imagination, L. (2012, August). Apache lucene 4. *In SIGIR 2012 workshop on open source information retrieval (p. 17).*

[11] Gormley, C., & Tong, Z. (2015). Elasticsearch: the definitive guide: a distributed real-time search and analytics engine. *" O'Reilly Media, Inc.".*

[12] gensim. (2021, March 25). PyPI. https://pypi.org/project/gensim/

[13] pytorch. (2021, March 25). PyPI. https://pypi.org/project/pytorch/

[14] scipy. (2021, March 25). PyPI. https://pypi.org/project/scipy/

[15] numpy. (2021, March 25). PyPI. https://pypi.org/project/numpy/

[16] scikit-learn. (2021, March 25). PyPI. https://pypi.org/project/scikit-learn/

[17] tensorflow. (2021, March 25). PyPI. https://pypi.org/project/tensorflow/

[18] Flask. (2021, March 25). PyPI. https://pypi.org/project/Flask/

[19] sentence-transformers. (2021, March 25). PyPI. https://pypi.org/project/sentence-transformers/