Универзитет у Београду

Електротехнички факултет

Logo

Description automatically generated

**Обрада природних језика**

Пројектни задатак

Ментор: Кандидати:

проф. др. Бошко Николић Бојан Селић - 2020/3185

др. Вук Батановић Ђорђе Ојданић - 2020/3258

Милисав Ковачевић 2020/3300

Београд, август 2021.

**Садржај**

[1. Увод 3](#_Toc81347699)

[2. Прикупљање података (прва фаза) 4](#_Toc81347700)

[3. Анотирање података (друга фаза) 6](#_Toc81347701)

[3.1 Анализа података 9](#_Toc81347702)

[4. Обрада података (Четврта фаза) 15](#_Toc81347703)

[4.1 Претпроцесирање података 15](#_Toc81347704)

[4.1.1 Без претпроцесирања 15](#_Toc81347705)

[4.1.2 Lower casing 15](#_Toc81347706)

[4.1.3 Бинаризација вредности *bag-of-words* одлика 15](#_Toc81347707)

[4.1.4 Фреквенцијско филтрирања речи 16](#_Toc81347708)

[4.1.5 TF, TF-IDF 16](#_Toc81347709)

[4.1.6 Филтрирање стоп-речи и стемовање речи 16](#_Toc81347710)

[4.1.7 Биграми и триграми 17](#_Toc81347711)

[4.2 Рангирање 17](#_Toc81347712)

[4.3 Класификација 18](#_Toc81347713)

[4.3.1 Логистичка регресија 20](#_Toc81347714)

[4.3.2 SVM 20](#_Toc81347715)

[4.3.3 Naïve Bayes 21](#_Toc81347716)

[4.3.4 Резултати 22](#_Toc81347717)

# Увод

Пројектни задатак на предмету **Обрада природних језика** служи за примену теоријског знања на конкретном проблему. У оквиру овог пројектног задатка било је потребно прикупити, анотирати и истренирати класификатор коментара(на српском језику) написаних на програмском језику ***C#***.

За израду овог пројекта коришћене су различите технологије, пакети и библиотеке. Програмски језик коришћен за израду пројектног задатка је **Python**, a коришћене библиотеке су: **pandas**, **numpy**, **re**, **scikit-learn**, **nltk, joblib, request**. Примарни скуп података је добијен кроз *GitHub API,* који је јавно доступан, али такође одређен број коментара је добијен из школских пројеката одрађених на основним студијама од стране чланова нашег тима.

За прикупљање и анотирање је коришћено више алата, од којих су неки креирани у склопу пројекта(скрипте за дохватање података са GitHub-a, парсирање фајлова и лоцирање C# коментара унутар фајла). С обзиром да наша екипа броји 3 члана, поставком задатка је неопходно прикупити и анотирати 750 коментара.

Након прикупљања и анотирања података, прелазимо у трећу фазу која подразумева претпроцесирање анотираних података, тренирање класификатора и рангирање. Поставком задатка је захтевано да се искористе следеће технике претпроцесирања:

1. Нормализацију свих текстова на мала слова(*lowercasing*)
2. Бинаризацију вредности *bag-of-words* одлика
3. Фреквенцијско филтрирање речи
4. TF, IDF и TFIDF пондерисање
5. Филтрирање стоп-речи и стемовање речи
6. Коришћење биграма и триграма

После претпроцесирања коментара, поставком задатака је одређено да се примене следећи класификатори:

1. Мултиномијални Бајесов наивни класификатор
2. Логистичка регресија
3. Класификација методом потпорног вектора

Као други део последње фазе је разматран приступ заснован на рангирању, где се од статистичког модела очекује да спроведе рангирање сличности између задатог упита и свих елемената документационих коментара, тј. проналажење коментара најсличнијег задатом упиту.

# **Прикупљање података (прва фаза)**

Како би се подаци прикупили, потребно је било пронаћи фајлове са *C#* кодом, тачније фајлове са **.cs** екстензијом. Ово је било могуће одрадити на више начина, међутим у пројекту се одлучило за следећа два приступа. Први приступ је био да се користи *GitHub Search Code API*  који очекује да му се проследи претраживана фраза(или реч), а покушано је са фразама наведеним у пројектном задатку са ID-јевима 0-99. Није на тај начин постигнуто да добијемо велики број фајлова који садрже коментаре на српском језику које бисмо искористили касније, а низак проценат оних које бисмо могли анотирати са степеном сличности 2 или 3. Као мала модификација првог приступа покушано је да се користе другачији облици речи претпостављајући да *GitHub Search Code API*  не ради неке напредније технике приликом индексирања фајлова за речи на српском језику. Постигнути су бољи резултати, прикупљен већи број коментара овим пондерисањем фраза, али је примећен следећи проблем. Тај проблем је био да се из **.cs** фајлова добија мали број коментара или се уопште не добијају коментари на српском језику, тј. велики број фајлова нема коментаре на српском док мањи број фајлова има само понеки коментар. Разлог томе је зато што *GitHub Search Code API*  враћа фајлова из различитих репозиторијума где је успео да направи најбоље преклапање, а ти репозиторијуми припадају различитим корисницима, док је реално да ће се пре пронаћи већи број коментара ако бисмо прегледали све .cs фајлове у једном репозиторијуму код корисника који веома често документује своје методе и класе. Да бисмо решили тај проблем, коришћен је *GitHub Advanced Search UI* који даје одличан приказ по претраживаној речи(фрази) и програмском језику. На њиховом *UI* јасно могу да се примете репозиторијуми код којих постоји велика вероватноћа да се пронађе велики број коментара, тј. да су ти корисници највероватније студенти рачунарства који су и циљна група за тражење коментара. *GitHub Advanced Search UI* враћа мањи приказ одсечка фајла приказан на слици 2.1 због ког је фајл препознат као добар кандидат, а из тог приказа се може закључити да ли је корисников репозиторијум добар кандидат за даљу претрагу. Ако из приказа видимо да је корисник именовао варијабле, методе, репозиторијум, коментаре(ако се виде у приказу) на српском језику, забележимо корисничко име(*owner*) и име репозиторијума. Када су претходно наведена 2 податка прикупљени, потребно је искористити један *GitHub API*  позив за повлачење свих фајлова из траженог репозиторијума одређеног корисника. Када се добију сви фајлови, потребно је исфилтрирати фајлове са екстензијом .cs.

Са претходна два приступа су прикупљани фајлови са екстензијом .cs, а имплементација се налази у фајлу *github\_scraper.py*.

Када су фајлови прикупљени и сачувани на одређеној путањи, потребно је било покренути скрипту *comment\_parser.py*. Ова скрипта за сваки фајл са екстензијом .cs креира посебан фајл са истим именом само са екстензијом **.csv**. У том новонасталом фајлу су исписани сви коментари из одговарајућег фајла са екстензијом .cs и приказано на којој линији у фајлу се налазе, због лакше претраге.

Поред наведеног приступа прикупљања фајлова, користили смо и ручно претраживање репозиторијума у *GitHub Advanced Search UI.*

Након што је парсирање коментара завршено и верификовано, коментари су чувани у посебним фајловима заједно са програмским кодом који описују(функцију, класу или модул) одвојени једним празним редом, а ти фајлови су сачувани у ***/resouces/file-annotations*** директоријумуунутар пројекта. Такође поред тога,сачувани су у заједнички Еxcel фајл(на *Microsoft Office 365* дељен између свих чланова тима).

Graphical user interface, application

Description automatically generated

Слика 2.1 – Github Advanced Search UI

# Анотирање података (друга фаза)

Након прикупљања података на горенаведене начине, неопходно је сваком прикупљеном коментару и блоку кода доделити релевантност између пронађеног програмског кода који коментар описује и одабране претраживане фразе из скупа приложеног у поставци пројектног задатка. На слици 3.1 може се видети принцип анотирања.

Graphical user interface, application, table

Description automatically generated

Слика 3.1 – Дељени excel фајл за анотацију

Са слике се може видети да *QueryText* и *SimilarityScore* колонe садрже листу вредности. Договором између чланова тима је усвојено да уколико посматрани програмски код(функција, класа или модул) има неки степен сличности са посматраним фразама(различиту од 0) да се све фразе наведу као листа чији су елементи одвојени знаком **,** или **#**. Колона *SimilarityScore* садржи такође листу идентичне дужине, где сваки елемент представља степен сличности одговарајуће фразе са посматраним програмским кодом.Уколико програмски код нема сличности ни са једном фразом, потребно је у обе колоне уписати вредност 0.

Сваки ред унутар овог скупа података је било неопходно ручно обрадити, по спецификацији пројектног задатка где је наведено да се сличност може анотирати следећим вредностима:

* 3 – потпуно подударање
* 2 – јако подударање
* 1 – слабо подударање
* 0 – потпуна ирелевантност

Након што су колоне *QueryText* и *SimilarityScore* попуњене, унутар скрипте *read\_excel\_speedy.py* врши се трансформација прикупљених 750 редова у дељеном excel фајлу,креирајући два нова фајла *output\_no\_similarity\_score.txt* I *output\_similarity\_score.csv*.

Фајлови:

**оutput\_no\_similarity\_score.txt** – приказује тражени садржај (ProgrammingLanguage, RepoDescription, SourceDescription, PairID, CommentText) за сваки ред из дељеног excel фајла

***output\_similarity\_score.csv*** – приказује тражени садржај(*ProgrammingLanguage, PairID, QueryText, CommentText, SimilarityScore*) где се за сваки ред из дељеног excel фајла креира нових 99 у овом фајлу.Прво се креира број редова једнак дужини листе за посматрани ред где је сличност већа од 0, а онда се за све остале фразе креирају нови редови где је сличност између фразе и посматраног програмског кода једнака 0. На тај начин добијен је фајл који садржи 74250 редова који ће се даље користити за претпроцесирање, рангирање и тренирање класификатора.

Фаза анотирања је углавном текла паралелно са првом фазом. Члан тима након што прикупи мањи скуп коментара на српском језику на начине описане у претходном поглављу, одлучивао је се по свом личном избору или у мањим консултацијама са остатком тима о сличности пронађеног програмског кода и фразе.

Приликом анотирања најтеже је било донети одлуку за оцене 1 и 2 из разлога што код њих немамо потпуно подударање или одступање већ треба нашом проценом да утврдимо сличност одговарајућем упиту посматрајући повратне вредности функције,сврху функције и колики проценат кода функције се преклапа са одговарајућом фразом.

Један карактеристичан пример оцењивања двојком се налази у примеру са *PairId-*ијем *bselic-comment-30* у коме се налази функција чији комплетан код креира речник од листе стрингова и тај речник је повратна вредност. Упит за који смо оценили са оценом 2 је груписање по броју.

Други карактеристичан пример оцењивања јединицом се налази у примеру са *PairId-*ијем *bselic-comment-74* у којој се налази void функција која, између осталог, генерише случајне бројеве, али ове генерисане бројеве користи у неку другу сврху.

Унутар тима је договорено да се коментари из фајлова не мењају ни на који начин, тј. да се задржи њихов оригинални облик из фајла, а да се у наредној фази препроцесирања пре него што се примене тражене технике из поставке задатка одраде неке стандардне технике чишћења текста, као што су:

1. Спајање више узастопних *whitespace-*ова у један
2. Брисање знакова интерпункције
3. Брисање бројева и специјалних знакова(задржавање само слова)
4. Спајање коментара у више линија у један коментар

10 % података је искоришћено за рачунање сличности анотације између чланова тима, односно сагласност анотатора. С обзиром да се ово десило након анотирања оригиналног скупа података, где су анотатори разменили мишљења и успоставили неки образац за анотирање, може се рећи да добијени резултати сличности анотације нису једнаки реалним резултатима који би се могли добити без претходних договора. Из другог угла, ти претходни договори око анотирања су били значајни како би класификатор добио што прецизније податке.

У табели 3.1 се могу видети резултати степена сличности између свака два анотатора из тима и групни бинарни степен подударности .

|  |  |
| --- | --- |
| Милисав Ковачевић – Бојан Селић | 94.63555555555556 % |
| Ђорђе Ојданић – Бојан Селић | 94.18222222222222 % |
| Ђорђе Ојданић – Милисав Ковачевић | 91.08444444444444% |
| Групни бинарни степен подударности | 88.88% |

Tabela 3.1 – сличност анотирања

Имплементација сличности анотирања се налази у фајлу *annotation-similarity.py ,* a излазни фајл је  **shared\_programming\_comments\_annotation\_aggregated** унутар директоријума **/resources/annotation-similariy**.

## Анализа података

У наставку ће бити приказана кратка анализа прикупљених података.

1. *Дистрибуција SimilarityScore-a* – као што је већ напоменуто укупан број коментара је 74250, од тога су већински вредности 0 за *SimilarityScore,* па је од интереса приказати дистрибуцију ненултих вредности за *SimilarityScore.* То је приказано на слици 3.2.

*Chart, bar chart

Description automatically generated*

Слика 3.2 – дистрибуција SimilarityScore-a

1. Проценат прикупљених података по анотатору је приказан на слици 3.3

Chart, pie chart

Description automatically generated

Слика 3.3 – проценат прикупљених података по анотатору

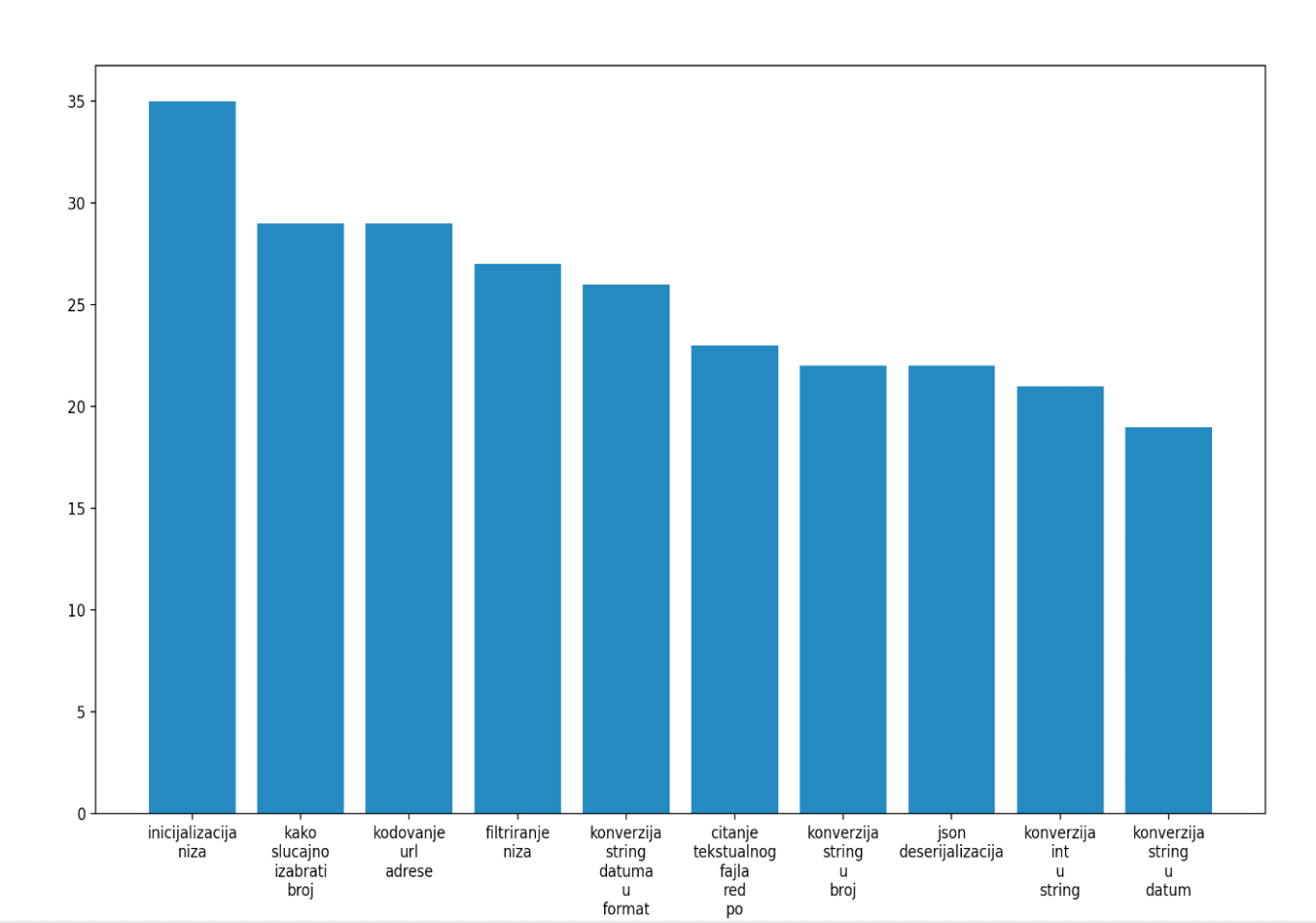
1. Проценат прикупљених вредности за сваку вредност *SimilarityScore-a* по анотатору. У табели 3.2 је приказана дистрибуција за све три вредности.

|  |  |  |
| --- | --- | --- |
| Chart, pie chart  Description automatically generated  1 | Chart, pie chart  Description automatically generated  2 | Chart, pie chart  Description automatically generated  3 |

Tabela 3.2 – проценат прикупљених вредности за сваку вредност SimilarityScore-a по анотатору.

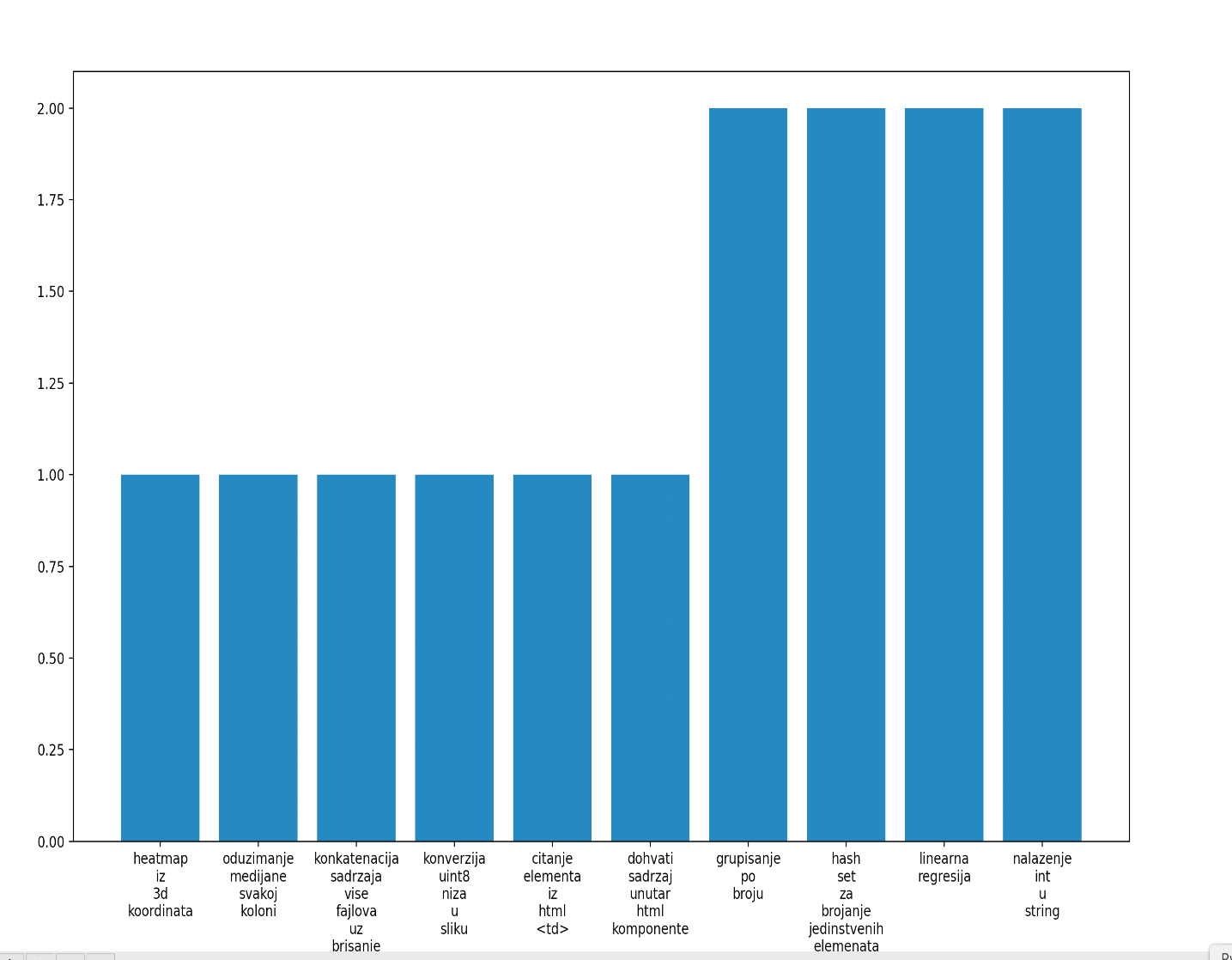
На сликама у табели 3.2 видимо да постоје разлике у заступљености оцена код анотатора, разлог томе је што смо распоредили да неки траже што квалитетније кодове ручно који би одговарали са већом оценом сличности, док за то време неко користи скрипту *github\_scraper.py* за прикупљања што већег броја коментара.

1. Приказ 10 најчешће анотираних фраза и број њихових ненултих анотација је дат на слици 3.4



Слика 3.4 – број ненутих анотација за 10 највише анотираних фраза

1. Приказ 10 најређе анотираних фраза и број њихових ненултих анотација је дат на слици 3.5



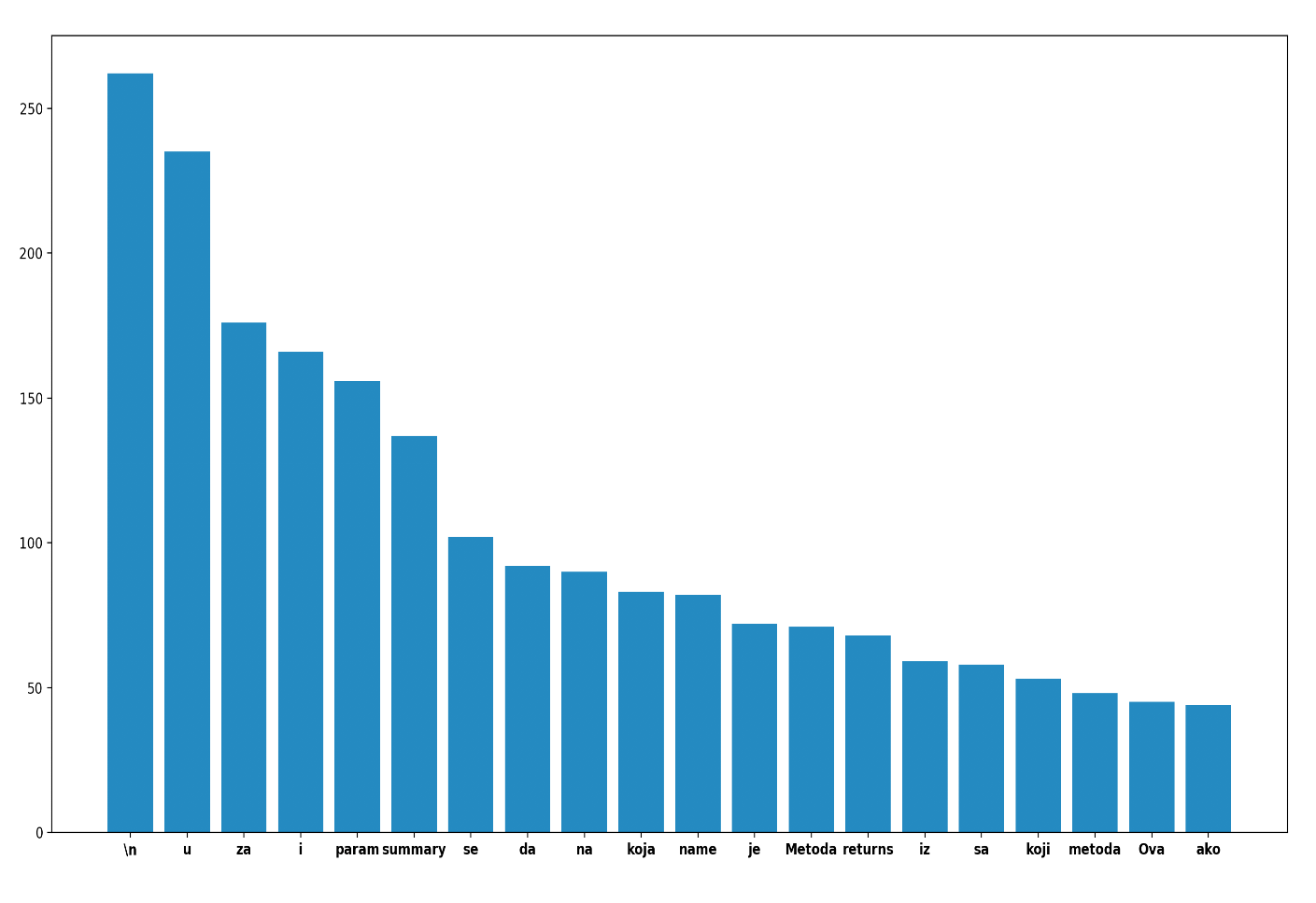
Слика 3.5 – број ненутих анотација за 10 најмање анотираних фраза

Поред упита на слици 3.5 има и оних који се уопште нису појављивали и они се налазе у следећој листи упита:

1. Подешавање радног директоријума
2. Socket recv timeout
3. K means кластеровање
4. Парсирање бинарног фајла у custom класу
5. Nelder mead оптимизацију
6. Постављање атрибута фајла на сакривен
7. Исписивање информација о моделу
8. Подвлачење текста у label widget
9. Мемоизација на диск – константна мемоизација

Разлог зашто одређени упити имају мање од 5 анотација већих од 0 је то што су акције које ти упити описују прилагођеније и учесталије у неком другом технолошком stack-у(као нпр. ML,Deep learning,NLP,Mobile инжењеринг...).

1. Toп 20 најфреквентнијих речи у коментарима је приказан на слици 3.6.



Слика 3.6 – топ 20 најфреквентнијих речи у корпусу коментара

Може се видети да у непретпроцесираном тексту су најфреквентније речи уствари везници, предлози и речце.

1. Приказ фреквентности свих речи могу се видети на *WordCloud* слици 3.7.

A picture containing qr code

Description automatically generated

Слика 3.7 – WordCloud репрезентација фреквентности речи у коментарима

1. Приказ опсега фреквентности речи у коментарима је приказан на слици 3.8

Chart, histogram

Description automatically generated

Слика 3.8 – опсег фреквентности речи у коментарима

Са слике изнад можемо закључити да се огромна већина речи појављује мање од 5 пута ако посматрамо непретпроцесирани корпус.

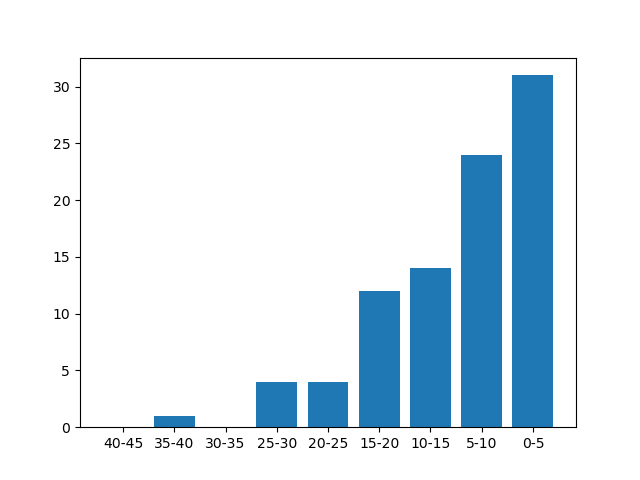
1. Просечна дужина коментара(број карактера) и број речи коментара за сваки анотациони тип је приказана на слици 3.9.

Chart, bar chart

Description automatically generated

Слика 3.9 – просечан број карактера и речи унутар коментара по анотационом типу

На слици 3.10 је графички приказано како су упити распоређени у распонима.



Слика 3.10 Приказ броја упита по оспегу појављивања

# Обрада података (Четврта фаза)

У овом поглављу је детаљно описана фаза претпроцесирања података, тренирање класификатора, рангирање и евалуација квалитета за класификаторе и рангирање.

## Претпроцесирање података

У поставци пројектног задатка је захтевана примена више различитих типова претпроцесирања. Свака врста претпроцесирања је примењивана и на сваки класификатор, а такође и на модел за рангирање.

### Без претпроцесирања

У овом типу претпроцесирања примењују се само брисање *null* вредности, брисање *html* тагова, брисање непотребних *whitespace*-ова. Наведена брисања се примењују код свих техника претпроцесирања текста. Резултати ове технике нам служи као референтна тачка за посматрање утицаја других техника на резултате класификације и рангирања.

### Lower casing

Као што је наведено, увек се примењују брисање null вредности, брисање *html* тагова и брисање непотребних *whitespace*-ова. Поред овога, сва велика слова се претварају у мала слова.

### Бинаризација вредности *bag-of-words* одлика

Свака од техника након примене над текстом се претвара у *bag-of-words* облик(осим *TF* и *TF-IDF*) које саме по себи након примене имају *bag-of-words* облик. Идеја бинаризације одлика је да се број појављивања речи кодира са 1(ако се реч појављивала) и 0 ако се реч није појављивала. Цео *bag-of-words* не значи да ће бити попуњен само вредностима 0 и 1, већ ће та вредност можда бити пондерисана са *IDF* фактором у зависности од фреквентности посматране речи у целом корпусу. То је постигнуто постављањем параметара **binary** = **True** u **CountVectorizer** интерфејсу(*sklearn* пакет).

### Фреквенцијско филтрирања речи

Као што је већ напоменуто, свака техника као излаз даје *bag-of-words* облик. Фреквенцијско филтрирање додатно брише речи(колоне) које су се или превише или премало појављивале у целом корпусу. Одлучено је да буду избачене речи које су се појављивале више од 97% и мање од 3%.То је постигнуто постављањем параметара min\_df=0.03, max\_df=0.97 у **CountVectorizer** интерфејсу(*sklearn* пакет).

### TF, TF-IDF

Обе технике претпроцесирања укључују бројање речи у оквиру улазног скупа података. *TF(Term Frequency)* подразумева да се код улазних података преброје посматране речи. *IDF(Inverse Document Frequency)* за разлике од *TF* у обзир узима колико се одређене речи појављују у остатку корпуса па утицај тих речи смањује множећи претходно срачунати *TF* фактор са израчунатим фактором *IDF.* У пројекту су приказани резултати за случајеве када је *IDF* искључено и са укљученим *IDF(* постављањем/искључивањем параметра use\_idf у **CountVectorizer** интерфејсу).

Треба напоменути да је за рачунање IDF фактора потребно разматрати само тренинг скуп података како нам тестни део података не би учествовао у формирању вредности IDF фактора приликом тренирања. Модел не треба да има никакву додатну информацију о тестном скупу у процесу тренирања.

### Филтрирање стоп-речи и стемовање речи

Као и претходно објашњене технике излаз ове технике ће бити у *bag-of-words* облику. Пре трансформације у *bag-of-words* облик, искључене су речи које представљају речи без битног значења, а то су обично речице, везници, предлози… Као листа стоп речи коришћена је листа пронађена на овој адреси <https://github.com/Xangis/extra-stopwords/blob/master/serbian> која је проширена са неколико речи. Коначна коришћена листа се налази у *stopwords.txt* фајлу у пројекту.

Након примене филтрирања стоп-речи, примењена је техника стемовања. Она подразумева довођења сличних речи који су у различитим облицима на заједнички облик. У ову сврху коришћен је стемер наведен у пројектном задатку, стемер Љубешића и Панџића. Ово је постигнуто тако што су одрађени следећи кораци за текст који смо желели да се стемује:

* Сачувати *SCStemmers.jar*  фајл у пројекту(овај корак се одради само једанпут)
* Исписати цео корпус речи(*dataframe*) у један фајл
* Извршити следећу команду :
* java -jar SCStemmers.jar StemmerID InputFile OutputFile
* Прочитати резултате из излазног фајла у корпус речи (*dataframe*)

Након претходно описаних корака потребно је добијени скуп података претворити у *bag-of-words* облик.

### Биграми и триграми

Претпроцесирање код ових техника је слично као претпроцесирање где се не врши никаква обрада текста. Једина разлика је што се код ових метода као улаз гледају *n-gram-и,* а у нашем случају су то биграми и триграми. *N-gram* је низ од *n* узастопних речи. Биграми представљају парове од по две узастопне речи, док су триграми аналогно тројке.

## Рангирање

Идеја код рангирања је да се креира модел који срачунавати сличност између сваког упита и свих коментара из скупа података. Срачунавање сличности се ради тако што се прво коментари и упити трансформишу у *bag-of-words* облик, па се упоређују њихови *bag-of-words* облици и забележи вредност косинусне сличности између посматране фразе и свих коментара. Резултујући модел можемо посматрати као мапу где су кључеви фразе, а вредности су листе које садрже вредности косинусне сличности између те фразе и свих коментара. Претходно описана мапа би садржала 99 кључева, а свака листа која представља вредност би садржала 74250 вредности, с обзиром да скуп података садржи 74250 коментара.

Након успешно креираног модела, евалуација се врши тако што се за сваки ред у скупу података који садржи *SimilarityScore* различит од 0 креира тестни скуп података, који садржи претходно поменути ред и насумично одабраних 99 коментара из оригиналног скупа чији додељени упит је идентичан оном који посматрамо и *SimilarityScore* свих њих јеједнак 0. Када је тестни скуп креиран, потребно је проверити колика је вредност косинусне сличности између посматране фразе и свих коментара из тестног скупа(те информације могу да се очитају из креираног модела(мапе)). Након што су очитане све вредности, потребно их је сортирати по опадајућем редоследу и закључити на којој позицији(индексу) се налази посматрани ред(ред који има *SimilarityScore* различит од 0). На основу тог индекса се израчуна реципрочни ранг. Након што се обраде сви редови који имају *SimilarityScore* различито од нуле из оригиналног скупа, потребно је срачунати средњи реципрочни ранг. Резултати за средњи реципрочни ранг по техникама су приказани у табели 4.1.

|  |  |
| --- | --- |
| Техника | Средњи реципрочни ранк(МRR) |
| Без претпроцесирања | 0.48036312318913055 |
| *lowercasing* | 0.4174589634606156 |
| стемовање + стоп-речи | 1.0 |
| биграми | 0.9604513064133017 |
| триграми | 0.997624703087886 |
| *TF* | 0.48610929925530394 |
| *TF/IDF* | 0.47765792152930536 |
| фреквенцијско филтрирање | 0.5420630734864574 |
| бинаризација *bag of words* | 0.4932575106127034 |

Tabela 4.1 – MRR вредности по техникама

## Класификација

Класификација представља процес категоризације података у две или више класа. Класификатори могу бити дискриминативни или генеративни. Под дискриминативне спадају они који покушавају директно да науче параметре сепарационе праве, површи или хипер-површи тако што минимизују неку критеријумску функцију и из ње добијемо оптималне параметре за сепарацију. И онда на основу рачунања наше хипотезе за одбирке добићемо класификоване податке. Са друге стране генеративни алгоритми су они који уче параметре расподела наших класа и онда на основу условних вероватноћа видимо којој класи одбирак највероватније припада. У нашем случају од дискриминативних алгоритама користили смо логистичку регресију и *SVM (Support vector machine),* а од генеративних *Naive bayes*. Сви модели су пројектовани у коду помоћу *scikit-learn* biblioteke. Метрика која је коришћена за мерење перформанси свих класификатора је *f1-score.* Приликом пројектовања класификатора једна од првих коју треба увести је регуларизација која представља заштиту од преобучавања, тај параметар представља хипер-параметар модела. Под хипер-параметре спадају сви они које мора пројектант да подешава ручно.

За све класификаторе смо имали захтев да упоредимо резултате када користимо *L1(LASSO)* и *L2(Ridge)* регуларизацију. Са теоретског становишта обе регуларизације имају неке своје предности, *L2* има аналитичко решење, а *L1* са друге стране може се показати преко Karush–Kuhn–Tucker-ове теореме и условне минимизације да ће нам *L1* регуларизација дати проређен модел који ће међу параметрима модела имати доста њих идентички једнаких нули.

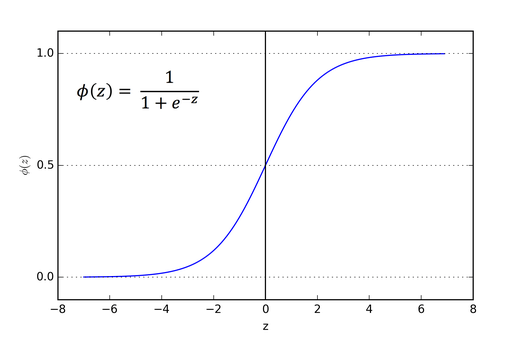
Техника коју користимо за тренирање модела и тражење оптималних вредности хипер-параметара је крос-валидација. Крос-валидацијом делимо наш тренинг скуп података на *fold-*ове, којих у нашем случају има 10, и у сваком проласку мењамо *fold* који користимо за валидацију, а све остале за тренирање. У задатку је коришћена стратификована крос-валидација која такође треба да обезбеди у сваком *fold-*у репрезентативни подскуп наше класе то јест да су пропорције заступљености класа из целог скупа приближне и у сваком *fold-*у.

Главни проблем оваквог скупа података за класификацију је био тај што су подаци изразито неизбалансирани. То се огледа у томе да класа 0 има заступљеност 99,9%, а остале класе заједно чине остатак. У таквом односу је неопходно пробати реализацију класификатора тако да неке примерке из ових мањинских класа успешно класификујемо. Очекивано је да прецизност на тим класама не буде висока, али исто тако на уштреб малог погоршања перформанси на већинској класи може се добити модел који може неке примерке мањинских класа да успешно идентификује. Решење које се у таквим ситуацијама може користити јесте балансирање класа, то значи да ћемо тежинским факторима више отежинити припаднике мањинских класа, а супротно урадити са већинском класом. На тај начин покушавамо да компензујемо разлику у броју података и нагласимо мањинске класе. Јако често овакви проблеми у пракси имају императив да класификација мањинских примерака буде значајно битнија од класификовања већинске класе. Код таквих примера покушаћемо више да пенализујемо грешку класификације код мањинске класе у односу на већинску. За поменуто балансирање класа треба дефинисати тежине које ћемо користити и из пробавања различитих вредности дошли смо до коначног избора. База коју смо користили је уграђена *python* функција *compute\_class\_weight* из *sklearn* пакета. Та функција враћа резултат од *n\_samples / (n\_classes \* np.bincount(y)),* што представља стандардан начин рачунања тежина класа(бр.одбирака/(број класа\*број података из те класе)).Онда смо такве вредности морали додатно да појачавамо на нашим мањинским класама јер са тим вредностима и даље нисмо могли ниједан одбирак из класа 2 и 3 да успешно класификујемо. Крајње тежине су дефинисане на следећи начин:

|  |
| --- |
|  |
| weights=compute\_class\_weight('balanced',['0','1','2','3'],y\_train)  class\_weights = {'0': weights[0], '1': weights[1] \* 12,  '2': weights[2] \* 12, '3':weights[3] \* 12  } |

### Логистичка регресија

Идеја код логистичке регресије је да нађемо параметре чији производ са предикторима пресликавамо помоћу логистичке(sigmoid) функције у распон [0,1] који нам је неопходан јер радимо класификацију(слика 4.3).

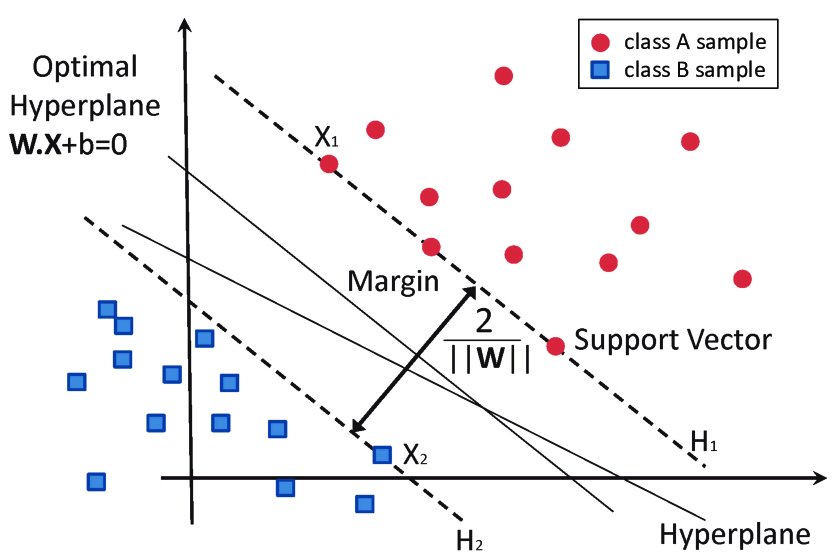


Слика 4.1 – Логистичка(сигмоид) функција

Код логистичке регресије постоје две опције за решавање *multiclass* проблема, jedna je pomoћу *one versus all* методе коју смо и ми користили, а друга је *softmax* метода. Изабрали смо прву методу јер је то услов програмског окружења како бисмо могли да користимо *multithreading* ради бржег извршавања тренирања модела. Логистичком регресијом се може вршити како линеарна тако и нелинеарна класификација увођењем полиномијалних одлика.

### SVM

*SVM(Support vector machine)* је други модел који смо пројектовали за класификацију наших података. Овај модел истo спада као што смо већ рекли у дискриминативне класификаторе који директно уче параметре функције која раздваја наше податке. Његова главна идеја је максимизација минималне геометријске маргине. Геометријска маргина представља растојање одбирка од сепарационе функције. Користећи Karush–Kuhn–Tucker-ову теорему можемо решити оптимизациони проблем којим добијамо параметре нашег модела и носеће векторе који имају то име јер они су ти од којих зависи наша сепарациона функција на основу које класификујемо податке.



Слика 4.2 – Пример SVM класификације за 2D случај

### Naïve Bayes

Овај класификатор спада у генеративне моделе који уче параметре расподела наших класа(математичко очекивање и варијансу) и на основу рачунања условних вероватноћа припадности одређеној класи коришћењем Бајесове формуле вероватноће закључујемо којој класи ћемо придружити наш одбирак. Претпоставка која се усваја код овог приступа је да су сви предиктори међусобно независни под условом да знамо из које класе долазе. На тај начин знатно олакшавамо израчунавање преко Бајесове формуле, а сама претпоставка се за доста реалних примера може сматрати потпуно адекватном. Из те претпоставке је овај класификатор и добио придев наивног.

### Резултати

У овом делу ћемо табеларно приказати добијене резултате за сва 3 класификатора трениране стратификованом крос-валидацијом и упоредити добијене вредности за L1 и L2 регуларизацију. Коришћена метрика је *f1-score.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Логистичка регресија | | SVM | | Naïve Bayes |
|  | L1 | L2 | L1 | L2 |  |
| without preprocessing | 0.95 | 0.86 | 0.95 | 0.94 | 0.45 |
| lowercasing | 0.94 | 0.85 | 0.94 | 0.94 | 0.5 |
| Binary bag-of-words | 0.95 | 0.86 | 0.95 | 0.94 | 0.46 |
| Stemming + stopwords | 0.93 | 0.85 | 0.94 | 0.93 | 0.51 |
| bigram | 0.97 | 0.9 | 0.97 | 0.97 | 0.43 |
| trigram | 0.97 | 0.9 | 0.98 | 0.96 | 0.31 |
| Frequency filtering | 0.93 | 0.84 | 0.93 | 0.91 | 0.98 |
| tf | 0.96 | 0.89 | 0.96 | 0.97 | 0.9 |
| Tf-idf | 0.93 | 0.64 | 0.95 | 0.95 | 0.9 |

Tabela 4.2 Табела резултата класификације

Из резултата прву ствар коју можемо да видимо јесте да Naïve Bayes класификатор даје генерално лошије резултате од осталих. Разлог томе јесте што генерално овакав модел није погодан за класификацију текстуалних података. Претпоставка на којој почива Naïve Bayes јесте да су предиктори међусобно независни што се у случају текста у облику реченица не може сматрати тачно. Очигледно је да када конструишемо реченице да речи нису међусобно независне јер знамо да речи које претходе одрећеној речи дефинитивно утичу на њу и појављивање неких речи сигурно има већу вероватноћу. Њихову међусобну условну независност бисмо могли да сматрамо да су реченице у ствари случајно набацан и генерисан скуп речи, али знамо да то није тако и из тог разлога се добијају лошији резултати. Код Naïve Bayes-а смо углавном успевали само да класификујемо одбирке из класа 0 и 1,остале две класе нису могле овим приступом да се препознају. Што се тиче резултата SVM и логистичке регресије видимо да су резултати доста бољи и резултати су прилично уједначени. Када гледамо поређења два приступа регуларизацији видимо да су резултати код SVM-а уједначени док се код логистичке регресије нешто бољи резултати добијају за L1 регуларизацију. У суштини смо код ових класификатора направили компензацију и добили већи број грешака за већинску класу 0 како бисмо добили неке успешна погађања за класу 2 и 3.