УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

ОБРАДА ПРИРОДНИХ ЈЕЗИКА

Пројектни задатак

Ментор:

проф. др Бошко Николић

др Вук Батановић

Кандидати:

Јован Стевановић - 2019/3154

Владимир Сивчев - 2019/3172

Предраг Митровић - 2019/3173

Матија Лукић - 2019/3279

Београд, јун 2020.

САДРЖАЈ

1.Увод

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

За израду овог рада било је потребно удружити неколико различитих технологија. Примарни скуп података је добијен кроз *API GitHub-a*, који је јавно доступан. *GitHub* тренутно представља најпопуларнији алат за верзионисање пројеката различитих величина. Кроз *API* може се приступи свим пројектима који су *open-source* статуса.

За прикуплање и анотирање података коришћено је неколико различитих алата. Наиме, неки од ових алата су специјално израђени као део пројекта, док су други већ постојећи који на један или други начин помажу за израду ове две фазе. Поставком пројекта је неопходно скупити и анотирати 3000 коментара.

Након прикупљања и анотирања података, прелазимо у трећу фазу која подразумева препроцесирање анотираних података и тренирање класификатора. Поставком пројекта је захтевано да се искористе следеће технике препроцесирања:

- *Lower casing* - пребацивање свих коментара на мала слова,
- *TF, IDF, TF-IDF* – подразумевају три методе које се базирају на фреквенцији појаве речи у коментарима,
- *Stemmer and stop words* – процесирање коментара коришћењем штемера и стоп речи,
- *Frequency word filtering* – филтрирање речи коришћењем учестаности њихове појаве, и
- *Bigrams and trigrams* – уместо да се процесирање врши на основу сваке појединачне речи, у овом случају правимо биграме и триграме од коментара.

*П*осле претпроцесирања коментара, поставком задатка је одређено да се примене следећи класификатори:

- *Мултиномијални Бајесов наивни класификатор*,

- *Бернулијем наивни класификатор,*
- *Логистичка регресија, и*
- *Класификација методом потпорног вектора без кернела.*

Последња фаза је подељена на две потфазе. Прва потфаза подразумева да се сви коментари који описују неку функционалност рачунају да су истог типа. Друга потфаза подразумева да се функционални тип коментара дели на три категорије које ближе описују циљ коментара.

2. ПРИКУПЉАЊЕ ПОДАТАКА (I ФАЗА)

Као што је већ речено, као главни и једини извор података за израду овог пројекта коришћен је *GitHub*. Сви пронађени пројекти су *OpenSource* статуса и коришћењем *GitExtensions* алата, исти пројекти се клонирају како би сада били доступни на локалном рачунару где се само прикупљање обавља.

Како би се подаци прикупили, првенствено је било неопходно пронаћи репозиторијуме из којих би се коментари извукли. Ово је било могуће одрадити на више начина, међутим у пројекту је донешена следећа одлука. Наиме, првенствено је била извршена претрага на *Google* систему, где су били претраживани било какви пројекти *SQL* типа, или да у себи садрже модуле који се баве неким *SQL*-ом. Резултати овакве претраге су навели на наредне сајтове, где постоје референце на разне пројекте који укључују *SQL*, а који се налазе на *GitHub*-у.

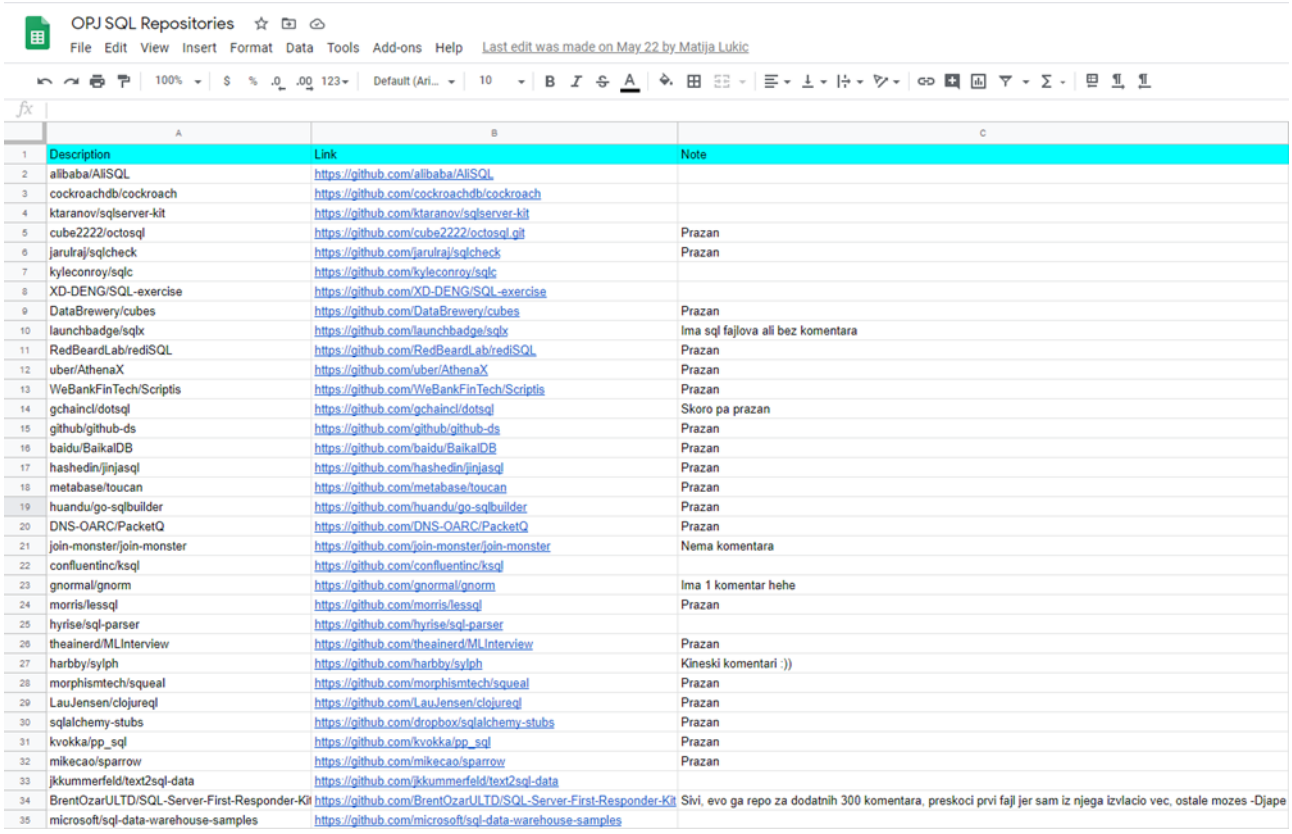
https://medium.com/issuehunt/50-top-projects-of-sql-on-github-in-2018-aabe0950a43a

https://awesomeopensource.com/projects/sql

https://awesomeopensource.com/projects/sql-server

Са претходних линкова је примарна претрага за репозиторијуме извршавана. Међутим, не садржи сваки референцирани репозиторијум оно што је у пројекту од интереса. Наиме, у овом тренутку се појављује неколико проблема.

С обзиром да је реч о тимском пројекту, била је неопходна синхронизација на то како ће се подаци прикупљати. Како би се избегло да више особа прикупљају податке са истог репозиторијума, направљен је *Google SpreadSheet* документ где су записивани до сада претраживани и разматрани репозиторијуми, уз додатне потенцијалне напомене за сваки репозиторијум. На слици 1 се може видети приказ овог документа, са првих 35 разматраних репозиторијума.



OPJ SQL Repositories

File Edit View Insert Format Data Tools Add-ons Help Last edit was made on May 22 by Matija Lukic

	A	B	C
1	Description	Link	Note
2	alibaba/AliSQL	https://github.com/alibaba/AliSQL	
3	cockroachdb/cockroach	https://github.com/cockroachdb/cockroach	
4	ktaranov/sqlserver-kit	https://github.com/ktaranov/sqlserver-kit	
5	cube2222/octosql	https://github.com/cube2222/octosql.git	Prazan
6	jarulraj/sqlcheck	https://github.com/jarulraj/sqlcheck	Prazan
7	kyleconroy/sqlc	https://github.com/kyleconroy/sqlc	
8	XD-DENG/SQL-exercise	https://github.com/XD-DENG/SQL-exercise	
9	DataBrewery/cubes	https://github.com/DataBrewery/cubes	Prazan
10	launchbadge/sqlx	https://github.com/launchbadge/sqlx	Ima sql fajlova ali bez komentara
11	RedBeardLab/rediSQL	https://github.com/RedBeardLab/rediSQL	Prazan
12	uber/AthenaX	https://github.com/uber/AthenaX	Prazan
13	WeBankFinTech/Scriptis	https://github.com/WeBankFinTech/Scriptis	Prazan
14	gchaincl/dotsql	https://github.com/gchaincl/dotsql	Skoro pa prazan
15	github/github-ds	https://github.com/github/github-ds	Prazan
16	baidu/BaikalDB	https://github.com/baidu/BaikalDB	Prazan
17	hashedin/jinjasql	https://github.com/hashedin/jinjasql	Prazan
18	metabase/toucan	https://github.com/metabase/toucan	Prazan
19	huandu/go-sqlbuilder	https://github.com/huandu/go-sqlbuilder	Prazan
20	DNS-OARC/PacketQ	https://github.com/DNS-OARC/PacketQ	Prazan
21	join-monster/join-monster	https://github.com/join-monster/join-monster	Nema komentara
22	confluentinc/ksql	https://github.com/confluentinc/ksql	
23	gnormal/gnorm	https://github.com/gnormal/gnorm	Ima 1 komentar hehe
24	morris/lessql	https://github.com/morris/lessql	Prazan
25	hyrise/sql-parser	https://github.com/hyrise/sql-parser	
26	theainerd/MLInterview	https://github.com/theainerd/MLInterview	Prazan
27	harbby/sylph	https://github.com/harbby/sylph	Kineski komentari :))
28	morphismtech/squeal	https://github.com/morphismtech/squeal	Prazan
29	LauJensen/clojureql	https://github.com/LauJensen/clojureql	Prazan
30	sqlalchemy-stubs	https://github.com/dropbox/sqlalchemy-stubs	Prazan
31	kvokka/pp_sql	https://github.com/kvokka/pp_sql	Prazan
32	mikecao/sparrow	https://github.com/mikecao/sparrow	Prazan
33	jkkummerfeld/text2sql-data	https://github.com/jkkummerfeld/text2sql-data	
34	BrentOzarULTD/SQL-Server-First-Responder-Kit	https://github.com/BrentOzarULTD/SQL-Server-First-Responder-Kit	Sivi, evo ga repo za dodatnih 300 komentara, preskoci prvi fajl jer sam iz njega izvlacio vec, ostale mozes -Djape
35	microsoft/sql-data-warehouse-samples	https://github.com/microsoft/sql-data-warehouse-samples	

Слика 1 – Синхронизациони документ за бирање репозиторијума

Као што се са слике 1 може видети, постоје одређени репозиторијуми који су празни. Такви репозиторијуми чак нису ни у мањини. Други проблем који је овде настао јесте потенцијална могућност на губитак у продуктивности (тако што би дати репозиторијуми улазили даље у наредне кораке процеса у пројекту, а да се на крају испостави да је то било непотребно је су репозиторијуми празни – немају коментаре који су од интереса). За ово је искоришћена *GitHub* опција која омогућава брзо површно (људско) скенирање репозиторијума које помаже у доношењу одлуке да ли овакав репозиторијум треба даље разматрати или га аутоматски искључити и прећи на други.

Поменута *GitHub* опција јесте *Find File* која омогућава да се репозиторијум претражује по фајловима, а у пројекту је наравно од интереса било извршавати претрагу на „*.sql*“, чиме се ефективно добијају сви фајлови са екстензијом *.sql*. Сада, могуће је летимично проћи кроз неколико фајлова и у зависности од процене човека, донети одлуку да ли је репозиторијум вредан даљег процесирања, или га треба одбацити са евентуалном напоменом да је празан.

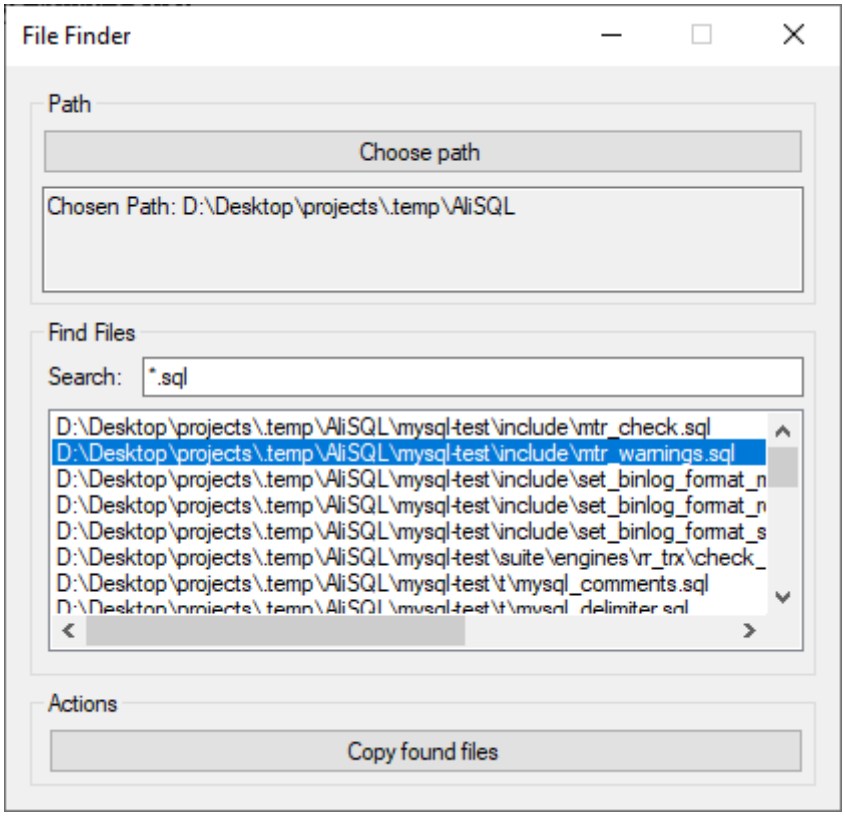
И празне репозиторијуме треба наводити у синхронизационом документу, тако да би наредне особе, уколико налете на исти, могле још брже да пређу на наредни репозиторијум, јер је тај већ био „процесиран“.

Када се пронађе репозиторијум који садржи коментаре од интереса, наредни корак у процесирању и прикупљању података јесте клонирање овог репозиторијума на локални рачунар како би било лакше обрадити репозиторијум даље. Наравно, и овај корак је могао другачије да тече, али је у пројекту донешена ова одлука из више разлога, а најбитнији јесте продуктивност, јер се користи већ направљени алат који има приступ одређеним деловима *GitHub API*-ја којем овај пројекат нема.

Већ је поменуто да је у ову сврху коришћен *GitExtensions* алат, мада било који други *Git* клијентски алат је потпуно равноправан овом. Једина опција која је овде од интереса јесте за клонирање датог репозиторијума.

Како сами пројекти могу имати произвољну структуру директоријума и датотека, потребно је на један или други начин доћи до *.sql* фајлова који потенцијално садрже потребне коментаре, али сада да аутоматизованији начин, а не ручном претрагом (ово је сада много лакше јер се фајлови налазе на локалном рачунару). Постоји више могућности како се ово може обавити, а у оквиру тима донета је пројектна одлука да се направе 2 нова алата који ће увећати продуктивност прикупљања података у овој фази.

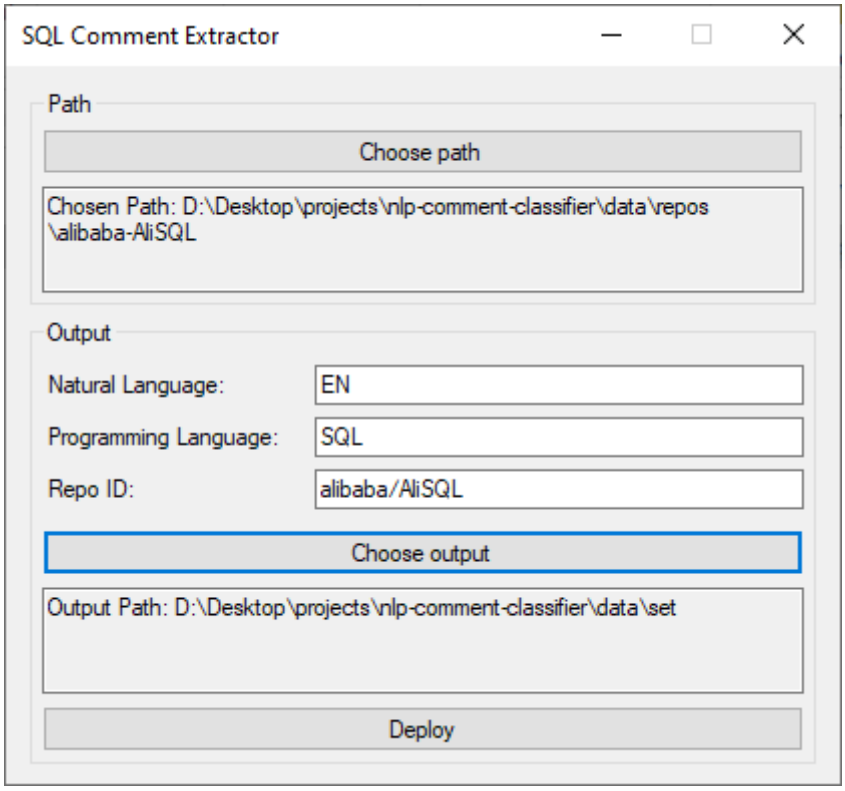
Наиме, први алат помаже у проласку кроз директоријуме клонираног пројекта и тражи фајлове по избору. На слици 2 се може видети изглед овог алата.



Слика 2 – Претраживач фајлова

Кроз овај алат могуће је одабрати основни фолдер у ком претрага започиње. Ово треба да буде основни (енгл. *Root*) директоријум клонираног пројекта у ком на даље треба пронаћи све потребне *.sql* фајлове. Уз помоћ оприје *Search* могуће је унети било какав израз по ком ће се претрага извршавати. Након притиска на тастер ентер, испод се излиставају сви пронађени фајлови из претраге. Такође, овај алат нуди могућности копирања пронађених фајлова на другу локацију притиском на опцију *Copy found files*. Сви пронађени фајлови ће се копирати на другу, изабрану локацију заједно са њиховим подфолдерима у оквирју самог пројекта. По конвенцији у оквиру тима, копирање се врши у оквиру репозиторијума самог текућег пројекта, тако да се сви фајлови из којих су коментари прикупљани такође налазе у оквиру овог репозиторијума.

Након копирања фајлова, неопходно је из њих извући коментаре и сложити их у један фајл по одређеном формату, који је описан у поставци пројекта. За ово је израђен други алат, чији се изглед може видети на слици 3.



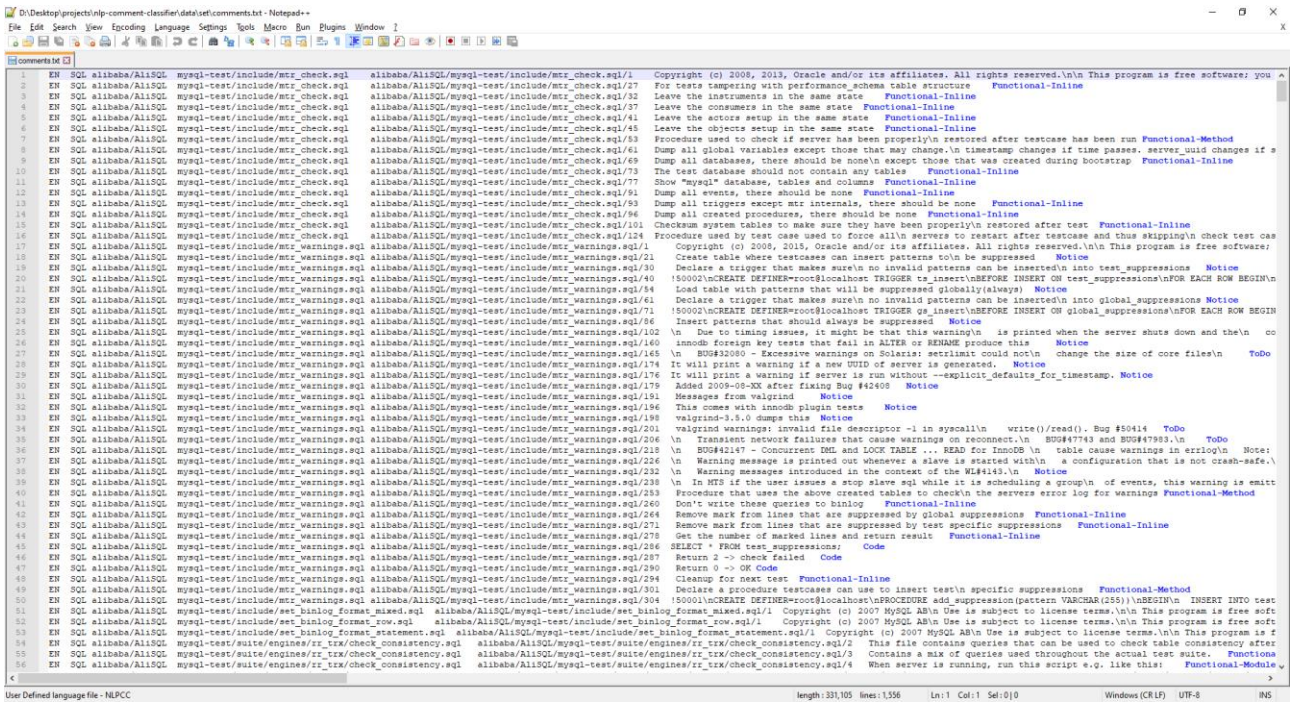
Слика 3 – Екстрактор коментара

Два наведена алата су тако пројектована да су компатибилна. Наиме, након што се фајлови копирају коришћењем првог алата, дестинациони фолдер копирања сада може да буде изворишни фолдер одакле се коментари извчаче, што се може подесити коришћењем опције *Choose path*. Након тога, треба евентуално подесити *Repo ID* тако да се уместо знака „-“ (конвенција усвојена у тиму пројекта) ставити знак „/“, како би сам *ID* био валидан. Потом, треба одабрати дестинациони фолдер где ће се направити два излазна фајла која ће садржади извучене коментаре из изворишта, по формату описаном у поставци пројекта. Наиме, уколико излазни фајлови већ постоје, алат само надовезује на претходни садржај тих фајлова (операција *append file*).

Притиском на опцију *Deploy*, алат пролази кроз све фајлове кроз читаву подструктуру датог изворишног фолдера, и у сваком пронађеном фајлу покушава да пронађе све врсте коментара који постоје у *.sql* језику.

3. АНОТИРАЊЕ ПОДАТАКА (II ФАЗА)

Након прикупљања података са горе описаним алатима, неопходно је анотирати којој класи припадају ови подаци. Овај посао је рађен ручно, и то коришћењем одређених окружења, као што је *VSCode, Nodepad++*, *Sublime* и слично. На слици 4 се може видети принцип анотирања у *Notepad++* окружењу.



Слика 4 – Анотирање у *Notepad++*

Са слике се може видети да су плавом бојом означене конкретне класе датих коментара по формату задатом у поставци пројекта. Наиме, сваки коментар је неопходно ручно обрадити на овај начин.

Ова фаза је донекле текла паралелно са фазом 2. Наиме, члан тима након што пронађе валидан репозиторијум и извезе коментаре описане у фази 1, такође треба и да прође кроз њих како би проверио да ли су потребне евентуалне дораде око коментара, као што је на пример, скупљање једнолинијских коментара нанизаних један иза другог у један коментар, јер уместо да је сам коментар написан у један вишелинијски коментар, он је био написан као више једнолинијских коментара, један за другим. У овом случају, потребно је скупити те коментаре и написати их као један вишелинијски коментар. У току овог пролаза могуће је паралелно и анотирати дати коментар.

Анотација сама по себи представља процес који је у одређеним ситуацијама једноставан и јасан, а такође у другим ситуацијама комплексан и не тако очигледан. Наиме, од класа које су описане у поставци пројекта, коментари типа *ToDo* и *Code* су они који су најједноставнији за уочавање и око њих уопште није било недоумица. Такође, коментари типа лиценца и ауторских права, информације о верзији, о аутору и сличних ствари су такође са лакоћом били разрешавани, а то су били коментари *General* класе.

Насупрот другим језицима где је то поприлично јасно, у *SQL* језику је тешко приметити коментаре класе *IDE*. Међутим, ипак су успешно били кроз прву фазу пронађени коментари који представљају шаблонске коментаре налик на *JavaDoc* анотатирање. Неколико примера око оваквих коментара:

```
name: GetAuthor :one
name: GetBook :one
name: DeleteBook :exec
name: BooksByTitleYear :many
```

Поред тога, постоје и још неки одређени коментари који би потенијално могли припадати класи *General* јер садрже информације о верзији и слично, међутим, у себи садрже име конкретне базе која је коришћена попут *MySQL* и слично, што означава да је овај коментар ипак био генерисан од стране коришћеног окружења, па је и сам коментар у том случају сврставан у класу *IDE*. Пример таквог коментара:

```
MySQL dump 10.13  Distrib 8.0.12, for osx10.14 (x86_64)\n Host: localhost  Database: cockroachtestdata \nServer version
```

Иако постоје одређене разлике у анотирању од стране сваког анотатора (што ће бити представљено касније), неке ствари су биле међусобно усаглашене током анотирања. На пример, донешена је пројектна одлука да се коментари који у себи садрже само линк ка одређеној *Web* страници сврстају у класу *Notice*. Наиме, разлог иза тога је био да обично линкови упућују програмера који гледа код ка неким додатним ресурсима који му помажу да лакше разуме о чему се у датом тренутку ради, или како би се нешто боље/сликовитије представило, објаснило и слично.

Класа *Notice* такође представља једну класу која у неким ситуацијама може да равноправно посматра као и класа *Functional-Inline,* па да постоји конфликт у томе која би се од те две класе користила. Наиме, ово је једна од ситуација коју сваки анотатор решава сам по себи, јер није био пронађен ни један разлог да се уведе одређени систем по ком би се одређена класа преферирала.

У одређеним ситуацијама, неки коментари су били дељени на 2 дела, тако да један део припада једној класи, а други другој.

Класа *Functional-Inline* у другим ситуацијама представља класу коју није толико тешко пропустити. Било какав опис кода који надолази након датог коментара јасно означава да тај коментар припада управо овој класи. Међутим, и овде постоје одређени изузеци. Поред тога, као што је већ речено, да ова класа може да се конфликтује са *Notice* класом, када постоји опис неког кода, ова класа се може конфликтовати са *Functional-Method* класом по овој логици. Међутим начин погледа како би се те две класе ефикасно разликовале јесте сам контекст. На пример, када би то био коментар у коду изнад дефиниције *Stored Procedure*, очигледно је да је реч о коментару типа *Functional-Method*. Међутим, овде може да постоји потенцијални проблем код класификатора касније. Наиме, уколико постоје два коментара који су веома слични, или чак можда идентични, а један се налази пре дефинисања одређене методе, док је други унутар ње, анотатор ће користити контекст како би одлучио који коментар припада једној од поменуте две класе. Међутим, сам класификатор нема такву информацију, па он види исти коментар (или сличан), сврстан у две различите класе (у тренутку када се ове класе разликују, а не када се све *Functional* класе посматрају као исте). Ово може да потенцијално умањи перформансе класификатора у трећој фази.

Слична дискусија може да се изведе и за *Functional-Module* класу. Међутим, с обзиром да *SQL* језик није објектно оријентисан (а ова класа углавном може да се користи за коментаре који описују читаву класу – читав модул), може се наслутити да таквих коментара и нема. Међутим, ово није истинито. Наиме, постоје одређени коментари који описују читав фајл, или можда део фајла који итекако припадају овој класи и никако ни не могу да се сврстају у неку другу, па је анотирање истих било једноставно, а такви коментари су се знатно разликовали од коментара других *Functional* класа, па ранија дискусија и не може тек тако да се примени на ову класу.

Поред сета за тренирање класификатора, додатно је још прикупљан одређени број коментара где је сваки члан тима засебно вршио анотацију, како би се срачунала сличност, односно сагласност анотатора. С обзиром да се ово десило након анотирања оригиналног сета, одређене одлуке које су заједно у тиму донете, потенцијално могу утицати на овај део, јер се анотатори, иако треба засебно, без комуникације да ово одраде, могу водити истим правилима као што су се водили кроз анотирање оригиналног сета. Међутим, ово ипак не представља проблем из наредног разлога. Циљ уношења оваквих пројектних одлука и јесте био да се анотатори синхронизују тако да анотација на крају буде што боља, што сагласнија, као да је један анотатор исти посао радио, како би се касније класификатору давали подаци који имају смисла. Али чак и са овим, и даље могу да постоје мале разлике у томе када 2 различита анотатора врше овај посао. Чак и са овим, у

издвојеном сету, сагласност између свака два анотатора у просеку износи негде око 90%, што указује на то да ипак постоји мали део анотирања који је субјективан и који може да утиче на класификатор. Међутим, ово не значи да је тај утицај лош. Наравно, није ни искључено да овај утицај може бити лош, али такође може и потенцијално позитивно да утиче на класификатор тако да не долази до преобучавања класификатора. У табели 1 се могу видети конкретни резултати рачунања сагласности.

Табела 1 – Сагласност анотатора	
Владимир Сивчев – Јован Стевановић	87.00%
Владимир Сивчев – Предраг Митровић	96.00%
Владимир Сивчев – Матија Лукић	95.33%
Јован Стевановић – Предраг Митровић	85.67%
Јован Стевановић – Матија Лукић	91.67%
Предраг Митровић – Матија Лукић	94.00%
Глобална сагласност	91.61%

У сваком случају, сам процес анотације је морао да тече ручно. Одређени коментари су такође били у потпуности уклањани из оригиналног и додатног сета, јер су или узроковали одређене проблеме, како за само тренирање касније, или једноставну неодлучност анотатора у томе којој класи би коментар припадао, или чак да ни једна класа није адекватна датом коментару. Таквих коментара је било веома мало, али су ипак постојали. У табели 2 су представљене расподеле коментара по класама.

Табела 2 – Расодела коментара по класама	
Functional-Inline	1259
Functional-Method	79
Functional-Module	31
ToDo	35
Notice	433
General	110
Code	838
IDE	235

4. Обрада података (III фаза)

У овом поглављу је детаљно описана фаза претпроцесирања података, тренирање класификатора и израчунавање квалитета класификатора.

4.1. Претпроцесирање података

Као што је наведено у уводу, у пројектној постави је захтевано више различитих типова претпроцесирања. С обзиром да је такође неопходно истренирати и евалуирати више класификатора, свака врста претпроцесирања је примењивана на сваки класификатор.

4.1.1. Без претпроцесирања

Код овог типа, извршава се све оно што се извршава и пре свих осталих фаза претпроцесирања. То подразумева да се уклоне све непотребне колоне, уклоне специјални карактери, *NA* вредности и дупликати.

Резултати евалуације без претпроцесирања служе као референтна тачка за посматрање утицаја који имају на перформансе сви остали типови претпроцесирања.

4.1.2. *Lower casing*

Као што је већ речено, сваки вид претпроцесирања укључује и основни вид претпроцесирања, а то је уклањање специјалних карактера, *NA* вредности и већ поменуто. Поред овога, наравно, сва слова у подацима се такође претварају у мала слова.

4.1.3. *TF*, *IDF*, *TF-IDF*

Овакве врсте претпроцесирања укључују бројање одређених речи у оквиру улазних података. Конкретно, *TF* (*Term Frequency*) подразумева да се код улазних података преброје конкретне речи које се користе. *IDF* (*Inverse Document Frequency*) за разлику од *TF* у обзир узима и то да се неке одређене речи у језику свакако често појављују па утицај таквих речи треба смањити. На пример, у енглеском језику, реч *the* се често појављује па њен утицај треба смањити. *TF-IDF* је само производ горе поменута два појма.

4.1.4. *Stemmer and stop words*

Штемовање речи је метода која се користи за нормализацију речи, односно да се приликом процесирања не гледа реч у целини него корен саме речи. У ову сврху је коришћен *Портеров штемер*.

Стоп речи подразумевају листу речи које у процесирању текста немају тежину. То су обично везници, узречице... Зато се оне уклањају како не би уносили нечистоће приликом даљег процесирања.

4.1.5. *Frequency word filtering*

Ова метода подразумева срачунавање фреквенције појаве сваке речи. Затим се врши филтрирање речи на основу тога колика има је појава у односу на средњу вредност.

4.1.6. *Bigrams and trigrams*

Препроцесирање је слично као код препроцесирања где се не врши обрада. Разлика ових метода јесте та да се сада као улаз гледају такозвани *n-gram*-и, који могу бити биграми и триграми. Конкретно, *n-gram* је низ од *n* узастопних речи. Отуда, биграми представљају парови од по две узастопне речи, док су триграми аналогно тројке.

4.2. Тренирање класификатора

У овом одељку су детаљно описани сви класификатори који су коришћени у обради података.

Приликом обучавања свих класификатора користи се 10-слојна стратификована унакрсна евалуација. Оригинални скуп података се првенствено дели на 10 слојева. Након тога, у свакој итерацији се узима један слој (*fold*), који се користи за тестирање и евалуацију, док се остали делови користе као подаци за обучавање. Пре него што се подаци за обучавање у ту сврху искористе, потребно је применити одређену врсту претпроцесирања (уколико се и једна ради), а затим наставити са обучавањем, и након тога са евалуацијом.

Поред тога, код *SVM* и *LR,* потребно је и оптимизовати хипер-параметар *C*. Ово се ради угњежденом унакрсном валидацијом. Наиме, овде је потребно радити и валидацију ради оптимизације хипер-параметра, и евалуацију. Код угњеждене унакрсне валидације, ток евалуације тече на исти начин као што је претходно описано, стим да се након овога преостали подаци поново деле на 5 унутрашњих слојева од којих се један користи за валидацију, тј. оптимизацију хипер-параметра *C*.

Код евалуације модела, као метрика је коришћена ф-мера. Наиме, тачност није погодна метрика у овом случају јер не постоји равномерна расподела класа у сакупљеним подацима. Стога потребно је користити друге мере, као што су прецизност и одзив, или у овом случају коришћена ф-мера која представља хармонијску средину претходне две.

4.2.1. Мултиномијални Наивни Бајесов класификатор

Мултиномијални Наивни Бајесов класификатор преставља представља класификатор који се често користи приликом класификације текста. Назив наивни потиче од тога што се не узима у обзир постојање завиности између речи у тексту, тј. наивно се уводи ова претпоставка, која иначе није тачна. Све речи се тумаче као одвојене, дискретне целине.

4.2.2. Бернулијев Наивни Бајесов класификатор

Бернулијев Наивни Бајесов класификатор функционише по истом принципу као и горе наведени Наивни Бајесова класификатор, са малом оптимизацијом у специјалним случајевима.

4.2.3. Логистичка регресија

Моделом логистичке регресије описујемо везу између предиктора који могу бити непрекидни, бинарни, категорички, и категоричке зависне променљиве. На пример зависна променљива може бити бинарна - на основу неких предиктора предвиђамо да ли ће се нешто десити или не. Оцењујемо заправо вероватноће припадања свакој категорији за дат скуп предиктора.

Тестирање је вршено са Л1 и Л2 функцијама грешке. Такође, код логистичке регресије смо имали и оптимизацију хиперпараметра *C*. Оптимизација је утицала да перформансе буду стабилније, тј. са мањом осцилацијом и у просеку за нијансу боље него без оптимизација.

4.2.4. Метод потпорних вектора без кренела

Метод потпорних вектора без кренела представља основну варијатну алгоритма. Без кернела, подразумева се да као фунцкију раздвајања узимамо линерну функцију. Кернел може бити готово било која функција.

Представља непробабалистички класификатор јер је излаз модела само класификациона одлика, а не и вероватно припарности класи.

Тестирање је вршено са Л1 и Л2 функцијама грешке, сумарно гледано. Такође, код методе потпорних вектора смо имали и оптимизацију хиперпараметра *C*. Оптимизација је

утицала да перформансе буду стабилније, тј. са мањом осцилацијом и у просеку боље него без оптимизација.

4.3. Резултати

У наставку су дате четири табеле, по 2 пара, са резултатима. Прве две табеле представљају резултате прве фазе где се користи једна класа за функционалне коментаре (прва табела од наведене две представља конкретне резултате, док се у другој налазе оптимизоване вредности *C* параметра), док друге две табеле приказују исто само за другу фазу, где сада постоје 3 различите класе за ове коментаре, као што је описано у поставци пројекта. По колонама се налазе конкретни коришћени класификатори, а то су:

- МНБК (Мултиномијални Наивни Бајесовски Класификатор),
- МБНБК (Мултиваријациони Бернулијев Наивни Бајесовски Класификатор)
- ЛР (Логистичка Регресија),
- МПВ (Метода Потпорних Вектора),

док су по редовима наведена конкретна коришћена претпроцесирања. Као што је већ речено, коришћена метрика је ф-мера, усредњена по слојевима.

	МНБК	МБНБК	ЛР (*L1*)	ЛР (*L2*)	МПВ (*L1*)	МПВ (*L2*)
Без ПП	86.03	71.87	86.23	88.00	87.40	88.82
Lower cs	87.11	70.50	85.92	87.57	87.73	89.50
TF	76.93	72.92	82.15	84.72	86.87	88.33
IDF	87.05	72.48	87.26	88.59	87.79	88.49
TF-IDF	80.39	71.79	84.05	85.25	88.91	89.55
Stemming	82.41	76.41	85.83	87.40	86.53	88.56
FreqFilter	82.10	75.20	86.5	88.20	86.20	88.90
Bigrams	80.10	73.20	83.2	85.10	83.60	85.60
Trigrams	76.50	69.65	81.23	84.31	82.21	84.96

	Логистичка Регресија	Потпорни Вектори
Без претпроцесирања	4.6	0.28
Lower casing	4.6	0.46
TF	24.2	24.4
IDF	20.2	22.24
TF-IDF	22.1	22.8
Stemming	12.2	12.8
Frequency Filtering	4.1	0.46
Bigrams	2.31	0.28
Trigrams	4.6	2.31

	МНБК	МБНБК	ЛР (*L1*)	ЛР (*L2*)	МПВ (*L1*)	МПВ (*L2*)
Без ПП	84.10	70.56	85.23	87.11	85.23	87.62
Lower cs	85.09	71.44	86.92	88.77	86.23	88.75
TF	75.23	70.96	81.95	83.63	85.51	88.33
IDF	86.01	71.51	85.36	86.59	85.79	87.52
TF-IDF	80.39	71.79	84.05	85.25	88.91	89.55
Stemming	80.39	73.20	84.33	86.40	84.88	87.64
FreqFilter	80.30	74.90	85.50	86.33	85.40	87.40
Bigrams	80.50	72.44	81.31	83.15	82.31	84.44
Trigrams	75.10	67.54	80.77	82.13	81.11	82.89

	Логистичка Регресија	Потпорни Вектори
Без претпроцесирања	4.3	0.36
Lower casing	4.3	0.25
TF	22.2	24.4
IDF	24.2	20.12
TF-IDF	20.1	21.65
Stemming	12.4	10.8
Frequency Filtering	2.1	0.39
Bigrams	2.31	0.36
Trigrams	4.3	2.31

Из наведених резултата може се видети да Мултиваријациони Бернулијев Наивни Бајесов Класификатор дефинитивно даје најслабије резултате од свих коришћених метода. Чак и као најгори класификатор, перформансе му ипак нису толико ниске.

Метода Потпорних Вектора се показала као најбоља од свих, мада Логистичкој Регресији уопште не бежи за много. Наиме, скоро да ове две методе дају исте перформансе. Овај коментар се такође односи на варијанту са *L2* функцијом регуларизације, која се показала као боља за мали део перформанси у односу на *L1*.

За разлику од осталих класификатора, Мултиномијални Наивни Бајесов Класификатор показује доста варијација у перформансама. У одређеним моментима, перформансе му скоро достижу перформансе ЛР и МПВ, док у неким моментима (са другим претпроцесирањима) има пад у перформансама скоро до МБНБК.

Што се самих метода претпроцесирања тиче, *IDF* и *TF-IDF* технике су се показале као најбоље, док је техника *Trigrams* показала доста лоше резултате. Поред тога, чак и техника *Lowercasing*, као једноставна техника за претпроцесирање показује сасвим добре резултате. У неким ситуацијама, *Stemming* и *Frequency Filtering* дају резултате који су међу најбољим (нпр. за ЛР).

Из наведеног се може видети да су успешно били прикупљени и анотирани подаци, а затим истренирани одређени класификатори и на крају је била успешно извршена евалуација модела.