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.*

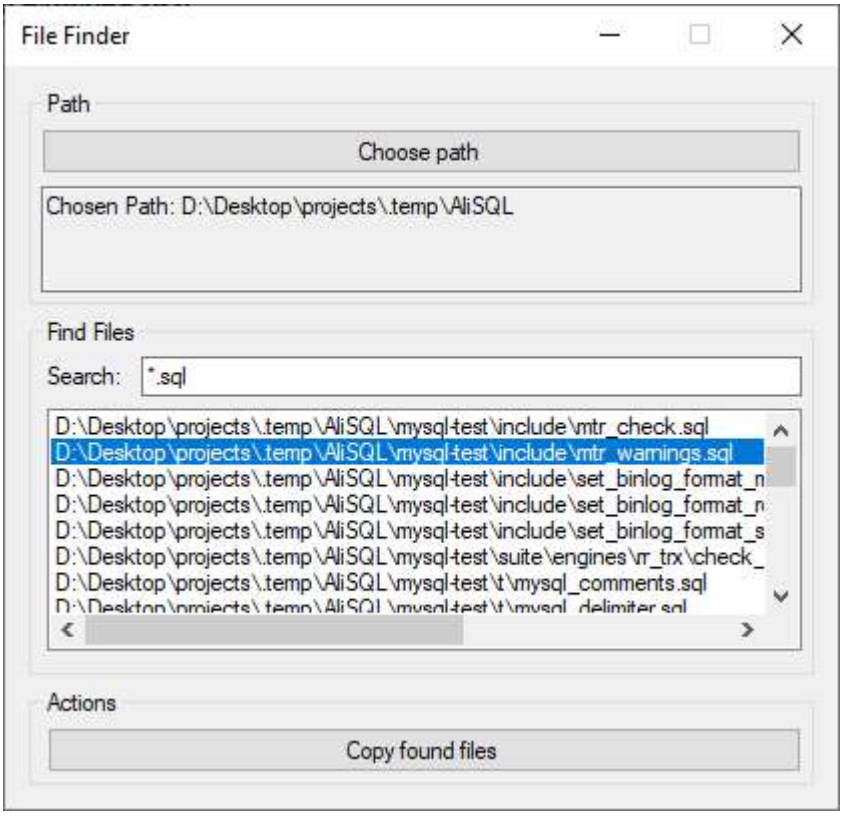
Последња фаза је подељена на две потфазе. Прва потфаза подразумева да се сви коментари који описују неку функционалност рачунају да су истог типа. Друга потфаза подразумева да се функционални тип коментара дели на три категорије које ближе описују циљ коментара.

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 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. 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 1 се може видети изглед овог алата.

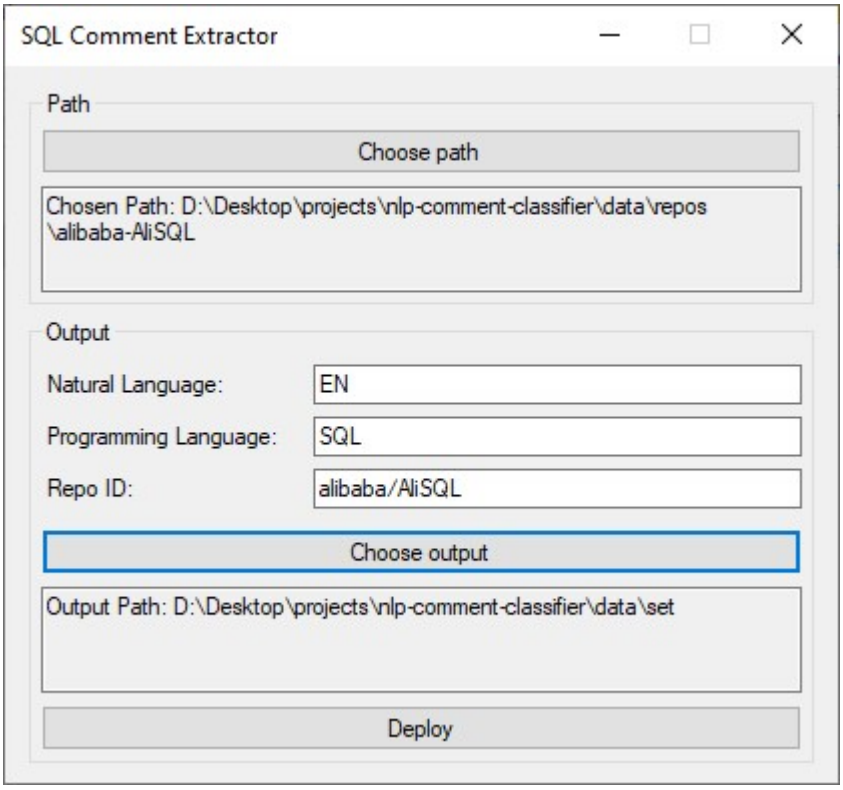


Слика 1 – Претраживач фајлова

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 опције *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 2.



Слика 2 – Екстрактор коментара

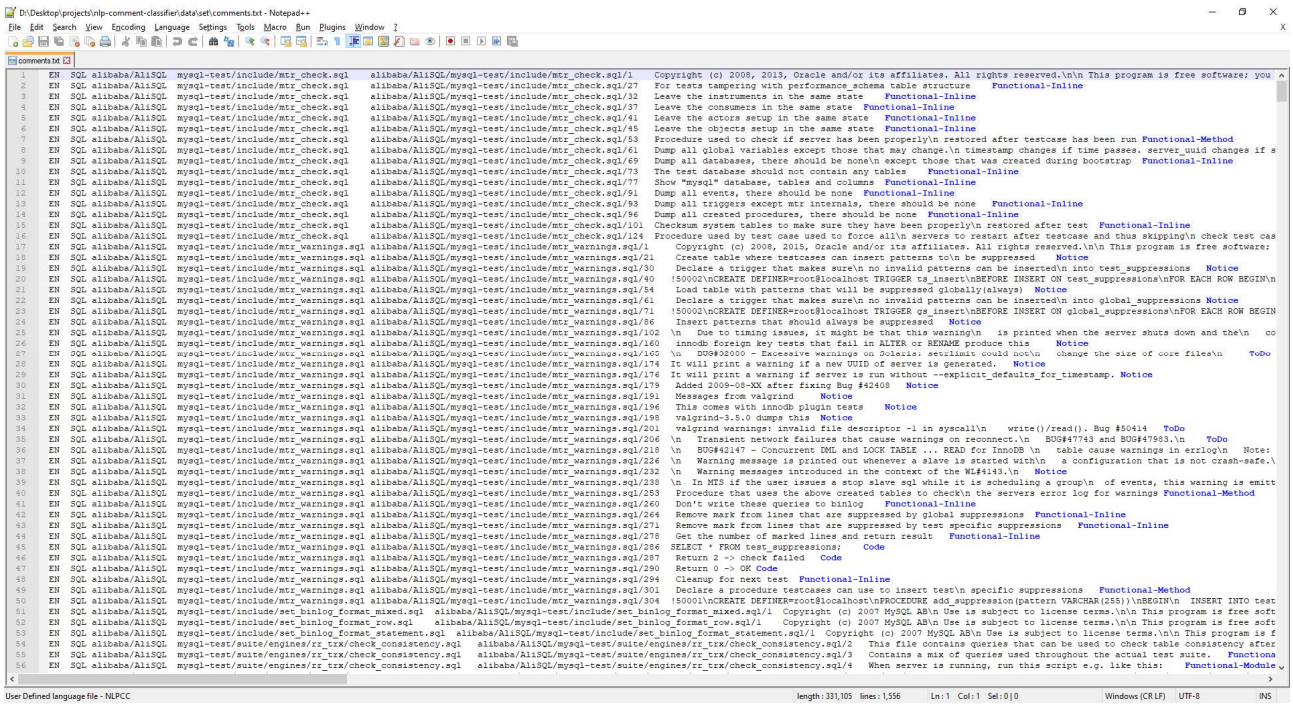
Два наведена алата су тако пројектована да су компатибилна. Наиме, након што се фајлови копирају коришћењем првог алата, дестинациони фолдер копирања сада може да буде изворишни фолдер одакле се коментари извчаче, што се може подесити коришћењем опције *Choose path*. Након тога, треба евентуално подесити *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 3 се може видети принцип анотирања у *Notepad++* окружењу.



Слика 3 – 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.

У овој фази, одређени коментари су такође били ручно на одређен начин процесирани. На пример, у неким пројектима су се налазили једнолинијски коментари заређани једно након другог, који су могли представљати један вишелинијски, али због начина на који су написани, алати у претходној фази их раздвајају навише појединачних. Стога, у овом случају, такви коментари су спајани ручно у један коментар.

Такође, неколицина коментара је могла да се сврста у више категорија. Такви коментари, иако представљају један, раздвајани су на два или више, како би се сваком делу доделила адекватна класа.

4. Обрада података (III фаза)

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

4.1. Препроцесирање података

Као што је наведено у уводу, у пројектној постави је захтевано више различитих типова препроцесирања.

4.1.1. Без препроцесирања

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

4.1.2. *Lower casing*

Подразумева иницијалну обраду, након које се сви коментари пребацују на мала слова.

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

Прва наведена метода подразумева да се као метрика узима учестаност појаве неког термина у тексту (***T**erm **F**requency*), друга метрика користи логаримску фунцкију учестаности појаве термина (***I**nverse **D**ocument **F**requency*), трећа представља комбинацију две горе наведене.

У сврху рачунања ових метрика користи се класа *TfIdfVectorizer* из пакета *sklearn.feature_extraction.text.*

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, већ се само поделе на унираме (први случај), само што се сада овде речи не деле на унираме, већ на биграме (две речи) и триграме (три речи).

4.2. Тренирање класификатора

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

Приликом обучавања свих класификатора се користи *10 слојна стратификована унакрсна валидација*. Која у случају мултиномијалног бајесовог наивног класификатора изгледа:

```
sss = StratifiedShuffleSplit(n_splits=10, test_size=0.1)
index = 1
for train_index, test_index in sss.split(x, data_frame['Type']):
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = data_frame['Type'][train_index], data_frame['Type'][test_index]

    mnb = MultinomialNB()
    mnb.fit(x_train, y_train)
    score = mnb.score(x_test, y_test)

    print("Score {}.: {:.3f}".format(index, score), end=" ")
    if index == 5:
        print()
    index += 1
```

Код логичке регресије и методе потпорних вектора користи се *угњеждена унакрсна валидација за оптимизацију хиперпараметара*. У случају логичке регресије то изгледа:

```
def optimize_c_parameter(data_frame):
    models_param = {
        'max_iter': [10000],
        'C': [1]
    }

    nested_cv_search = NestedCV(model=LogisticRegression(), params_grid=models_param,
                                outer_kfolds=5, inner_kfolds=5,
                                cv_options={'sqrt_of_score': True,
'randomized_search_iter': 30})
```

```
    x = preprocessing.get_data_set()
    nested_cv_search.fit(x, data_frame['Type'])

    optimized_c_value = np.mean(nested_cv_search.outer_scores)
    print("Optimized C: {:.3f}".format(optimized_c_value))
    compare_regularisation_functions(data_frame, 'l2', optimized_c_value)
```

4.2.1. Мултиномијални Бајесов наивни класификатор

Мултиномијални Бајесов наивни класификатор преставља представља класификатор који се често користи приликом класификације текстова. Назив наивни потиче од тога што се не узима у обзир постојање завиности између речи у тексту. Већ се све речи тумече као одвојене, дискретне, целине.

У сврху реализације ово задатка се користи следећи пакет:

```
from sklearn.naive_bayes import MultinomialNB
```

Тренирање се врши на следећи начин:

```
mnb = MultinomialNB()
mnb.fit(x_train, y_train)
```

Процена квалитета класификатора се извршава на следећи начин:

```
score = mnb.score(x_test, y_test)
```

4.2.2. Бернулијев наивни класификатор

Бернулијев наивни класификатор функционише по истом принципу као и горенаведени Бајесова наивни класификатор, са малом оптимизацијом у специјалним случајевима.

У сврху реализације ово задатка се користи следећи пакет:

```
from sklearn.naive_bayes import BernoulliNB
```

Тренирање се врши на следећи начин:

```
bnb = BernoulliNB()
bnb.fit(x_train, y_train)
```

Процена квалитета класификатора се извршава на следећи начин:

```
score = bnb.score(x_test, y_test)
```

4.2.3. Логичка 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.

У сврху реализације ово задатка се користи следећи пакет:

```
from sklearn.linear_model import LogisticRegression
```

Тренирање се врши на следећи начин:

```
lr = LogisticRegression(penalty=rf, C=c, solver=solver, max_iter=10000)
lr.fit(x_train, y_train)
```

Процена квалитета класификатора се извршава на следећи начин:

```
score = lr.score(x_test, y_test)
```

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.

У сврху реализације ово задатка се користи следећи пакет:

```
from sklearn.svm import LinearSVC
```

Тренирање се врши на следећи начин:

```
svc = LinearSVC(penalty=rf, C=c, dual=rf == 'l2', max_iter=15000)
svc.fit(x_train, y_train)
```

Процена квалитета класификатора се извршава на следећи начин:

```
score = svc.score(x_test, y_test)
```