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.

// ToDo – Ovde dodati nesto ukratko o tome kako je uradjena anotacija.

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 пртпроцесирања 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)

Примарни скуп података се налази у датотекама *UB_cs_authors.xlsx* и *UB_cs_papers_scopus.xlsx*. Прва датотека садржи 117 аутора научних радова из области рачунарства са Универзитета у Београду, док друга датотека садржи око 1200 научних радова поменутих аутора. Алат *Gephi 0.9.2.* за конструисање мреже захтева два улазна фајла. Ти фајлови су у *CSV* (*Comma separated values*) формату. Један садржи податке за креирање чворова мреже, а други садржи информације о везама између тих чворова. С' обзиром да су за одговарање на истраживачка питања биле потребне 2 мреже, секундарни скупови података су се налазили у 4 фајлова (по 2 за сваку мрежу).

Улазни подаци за *Java* апликацију су били из примарног скупа података, док су секундарни подаци добијени радом апликације и обрадом примарних података. За успешан рад *Java* апликације коришћења је пакет ***org.apache.poi***. Сваки фајл који је садржао секундарни скуп података који је задужен за описивање чворова мреже има обавезне колоне *Id* и *Label* (идентификациони број, и име чвора) као и произвољан број других колона које су у складу са захтевима за обраду те мреже. Фајл који служи за опис веза између чворова има четири обавезне колоне: *Source, Target, Type, Weight* (изворни и дестинациони идентификациони бројеви, усмереност веза у графу и тежина сваке везе).

Уз овај документ достављен је и изворни код *Java* апликације у коме се налази цела логика обраде примарног скупа података и припреме секундарних скупова података.

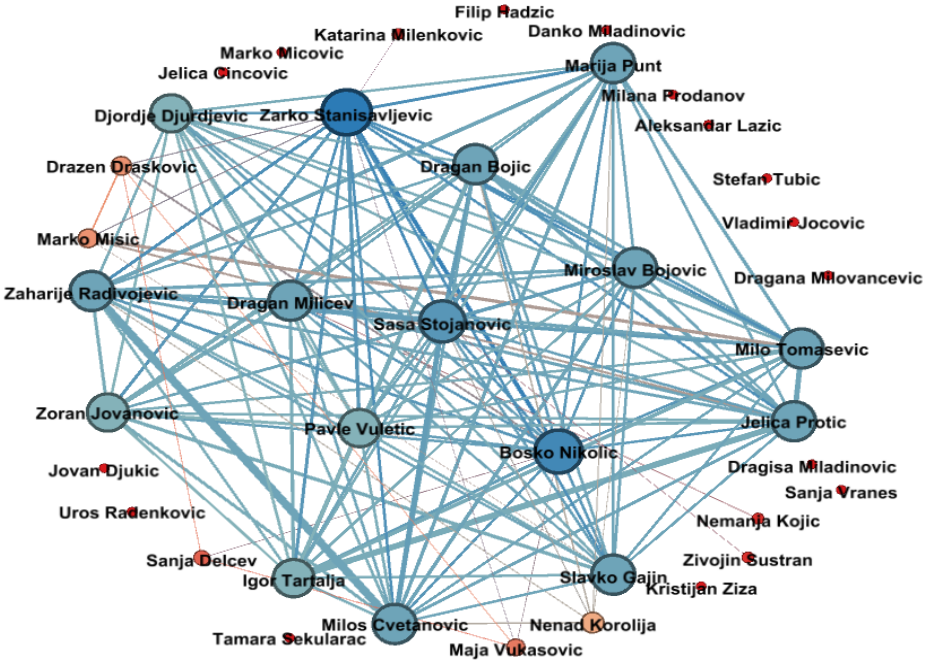
3. АНОТИРАЊЕ ПОДАТАКА (II ФАЗА)

За моделовање и графички приказ мреже коришћен је алат *Gephi 0.9.2*. На основу секундарних скупова података алат *Gephi 0.9.2* је конструисао тражене мреже. Након формирања мреже коришћене су функционалности алата за побољшавање графичког приказа мреже. Функционалности обухватају коришћење произвољног алгоритма за позиционирање чворова, бојење и подешавање величине чворова на основу атрибута, бојење и подешавање дебљине веза на основу атрибута, коришћење уграђених филтера за истицање чворова и група чворова од интереса.

За израду решења овог пројектног задатка направљене су две мреже: мрежа коаутора и мрежа научних часописа.

У оквиру мреже коаутора, аутори представљају чворове мреже, а везе између два чвора су успостављене уколико су два аутора учествовала заједно у изради бар једног научног рада.

У оквиру мреже научних часописа, часописи представљају чворове мреже, а везе између два чвора су успостављене уколико је један аутор писао и за један и за други часопис.



Слика 1. *Приказ мреже коаутора Електротехничког факултета.*

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)
```