Препознавање аутора на основу исечка текста уз помоћ репрезентације текста преко комбинације н-грама речи и н-грама карактера

Никола Радојчић 72м/22

21. август 2023.

Сажетак

Ово је проширење мог прећашњег рада [2]. Вадимо исечке из књига пет аутора. Потом их фитрирамо користећи *StringToWordVector* филтер у *Weka* алату са комбинацијом токенизатора н-граме речи у распону од 2 до 5, и знакова у распону од 1 до 3. На крају их класификујемо користећи *SMO* класификаторе и комбинујемо резултате методом меког гласања (*Soft voting system*).

Садржај

1 Увод

У прошлом раду [2] радио сам препознавање аутора на основу неког исечка и током израде тог рада сам наишао на рад колега из Копенхагена [1] који су комбиновали н-граме речи и карактера. Током израде мог прошлог рада ја сам наведен њиховим радом истестирао н-граме карактера и на крају су се показали много боље од мојих н-грама речи. Сврха овог рада је да покуша да споји н-граме речи и н-граме карактера по угледу на рад [1].

На препознавање аутора на основу неког исечка текста се може гледати као проблем класификације, где су нам аутори класе и треба да правилно класификујемо неки нови исечак на основу претходно виђених. Оно може бити веома корисно при утврђивању да ли је нешто плагијат. Такође може се користити за утврђивање аутора текста за који је аутор непознат. У апликацији коју сам правио за овај рад препознавање аутора је коришћено као помоћ током претраге исечака.

2 Обрада података

Као и у прошлом раду сви подаци у виду књига су добављени са сајта gutenberg.org. Већином сам користио исте ауторе као и у прошлом раду али да бих добио много балансиранији сет података свима сам додао много више књига које сам набавио са истог сајта. Један од аутора није имао довољно књига да бих са њима смањио разлику па сам га морао избацити и заменити са новим аутором чије сам књиге нашао на сајту. Све књиге су у формату текст фајлова па сам у *Javi* написао програм који вади исечке и ставља их у *.arff* фајл. Програм прво исчита сав текст у некој од књига, потом обрише првих 400 и последњих 500 линија текста пошто оне нису део књиге већ претежно заједнички за све књиге. Сада програм пролази кроз текст линију по линију и пази да не запише линије које не садрже слова јер то опет нису делови књига већ последица неког њиховог форматирања. Такође програм прескаче линије у којима се налази реч поглавље јер је то заједничко за све књиге. Када стигне до празног реда у књизи програм све линије текста од тог до прошлог празног реда сматра као исечак и записује ју у фајл заједно са информацијом који је аутор исечка. До сада наведену обраду текста сам имао и у прошлом раду али остаје много исечака који нису правилни јер нисам имао начин да их све прегледам због њихове велике количине.

2.1 Додаци обради исечака

Да бих могао боље да прегледам постојеће исечке користио сам *Lucene* библиотеку да индексирам *.arff* фајлове. Потом сам направио *backend* користећи *Spring* и *frontend* користећи *React* да бих могао да гледам резултате претраге у лаком формату. Ова нова апликација дозвољавала лаку претрагу кроз исечке. Да бих видео разлике у резултатима направио сам је тако да при сваком покретању изнова прави *.arff* фајлове и изнова их индексира. Последица овога је било да бих за сваку измену морао чекати више од минуте да се апликација подигне. Пошто сам требао да направим велик број честих измена морао сам оптимизовати апликацију. Оптимизацију сам постигао *multithreding-om*. Првобитно сам правио по једну нит за прављење тест и тренинг сета и једну нит за индексирање док је *Spring* радио на својој нити. Ово је постигло мала убрзања и апликација се дизала у року од педесетак секунди, што и даље није било довољно. Да бих постигао брже време одлучио сам да направим по две одвојене нити за сваког од аутора где би једна обрађивала књиге тог аутора за тренинг сет а друга за тест сет. Нове нити би у *.arff*

фајлове писале преко синхронизованих метода за писање у њих. Такође сам увео *multithreding* у индексирање где би креирао максималан број нити који процесор датог рачунара може да подржи у исто време. После ових оптимизација апликација се подизала у року од 24-26 секунди док су се *.arff* фајлови правили у року од 7-10 секунди. После оптимизација претраге користио сам апликацију да избацим погрешне исечке тако што бих гледао резултате неке претраге и правио измене логике које би избациле те исечке и потом поновном претрагом проверавао да ли сам успео. На овај начин сам успео да пронађем мане у мојој пређашњој логици.

Током израде прошлог рада мислио сам да свака књига има поглавље означено са речју поглавље великим словима што сам убрзо после израде апликације увидео да је погрешно. Анализирајући исечке направио сам регуларни израз који се поклапа са свим изразима који садрже реч поглавље написано великим или малим словима праћено са римским бројевима или двоцифреним арапским бројевим што је избацило сваку појаву речи поглавље која се не дешава као део текста књиге. Даљом анализом схватио сам да ми исечцима треба даља обрада па сам помоћу регуларних израза избацио и речи *page*, *section*, *volume* и *footnote* праћене са бројем. Такође сам избацио и појаве 21 специјалне речи које нису део текста књиге.

После ових промена добио сам два фајла који су садржали тренинг сет са 171602 инстанце и тест сет са 24204 инстанце то јест око 60000 додатних инстанци чак и после избацивања великог броја инстанци додатном обрадом.

3 Филтрирање

Да би радили са добијеним подацима сачињеним од текста исечка и класе аутора морамо текст репрезентовати на неки начин. Начин који ћемо користити је такозвана "*bag of words*" репрезентација. Репрезентацију текста сам радио преко *StringToWordVector* филтера у *Weka* библиотеци. У прошлом раду сам изанализирао велики број токенизатора а најбоље су се показали н-грами карактера у распону од 1 до 3 и н-грами речи у распону од 2 до 5 па сам одлучио да њих искомбинујем да бих добио жељену комбинацију. Да бих то одрадио направио сам два *StringToWordVector* филтера по један за сваки токенизатор и провукао оригиналне податке кроз њих, са спојеним тест и тренинг сетом у једно. Овде сам први пут достигао ограничења у ресурсима мог рачунара па првобитна идеја да пустим оба филтера у исто време на одвојеним нитим није успела па сам морао да их пуштам одвојено. Да бих штедео RAM после завршетка филрирања исписао бих резултате сваког од филтера у одвојени фајл. Потом бих комбиновао те резултате тако што бих у деловима читао из та два фајла и записивао атрибуте сваке инстанце у нову инстанцу, пазећи да не додам неки атрибут два пута ако постоји у оба резултата, потом бих ту инстанцу додао у нови сет. Нови сет бих потом опет записао у одвојени фајл у случају да апликација падне у међувремену. Добијени сет података је имао велики број атрибута па сам, због ограничења рачунара, провука добијени сет и кроз *AttributeSelection* филтер са инфо гаин евалуатором. После филтрирања имао сам три репрезентације текста преко н-грама речи, н-грама карактера и комбиновања претходна два.

4 Резултати

Да бих добио резултате користио сам *SMO* класификатор раздвојивши податке који су заједно филтрирани на тренинг и тест сетове тако што сам скинуо тест сет са краја сета података јер сам га ту и ставио током спајања. Зато што су подаци измењени заменом једног од аутора и избалансирани додавањем великог броја инстанци они се не могу поредити са оним из прошлог рада. Такође је обрада података сада много детаљнија.

Тренирање *SMO* класификатора над целим тест сетом са комбинованом репрезентацијом није било успешно. После 20 дана тренирања рачунар се сам угасио пре него што је стигао да заврши. Увидевши мањак ресурса мог рачунара одлучио сам да тренирам на 25% оригиналног тренинг сета. Резултати овог тренирања се могу видети у наставку.

Evaluation results:
Correctly Classified Instances 7280 30.0777 %
Incorrectly Classified Instances 16924 69.9223 %
Kappa statistic 0.1138
Mean absolute error 0.3041
Root mean squared error 0.4038
Relative absolute error 96.0856 %
Root relative squared error 101.5236 %
Total Number of Instances 24204
Training is complete in 93717.186 seconds

Као што се може приметити резултати нису били ни близу задовољавајући, па сам одлучио да променим метод на који класификује ауторе.

Идеја новог метода је у класификатору који се заснива на меком гласању (*Soft voting system*) од стране два класификатора једном тренираном на н-грамима [2,5] речи а другом на н-грамима [1,3] карактера. Овај класификатор узима вероватноће да дата инстанца припада некој класи од та два класификатора и помножи их са својим тежинама и сабере их, потом она класа која има највећу вероватноћу се узима за класу те инстанце.

У табели 1 можемо видети резултате нове репрезентације текста над 25% сета, одлучио сам да тренинг обавим над мањим делом сета јер је тренинг трајао дуго. Током тренинга коришћена је тежина 1 за оба класификатора па она није утицала на резултат.

	SMO
н-грам[2,5] речи	23.116
н-грам[1,3] карактера	22.4219
класификатор гласањем	48.0706

Табела 1: табела са процентима правилно класификованих инстанци

Из ове табеле се може увидети да класификатор добијен гласањем доноси убедљиво најбоље резултате.

4.1 Апликација

Као што сам већ рекао у циљу израде овог пројекта сам направио апликацију која исписује резултате и даје опцију за препознавање аутора и претрагу исечака потпомогнуту класификатором. Код апликације се налази на следећем линку github.com/Nikolar1/snippetBackend. Апликација је запакована у *docker* контејнер и лако се покреће. Због огромне количине времена потребне за филтрирање података и потом за тренирање класификатора апликација долази са постојећим моделима и већ филтрираним подацима.

5 Закључак

Када се сагледају резултати метод преко комбиноване репрезентације који сам намеравао да користим се показао, иако бољим од засебних репрезентација, и даље као не адекватан. Док нови метод који је сличнији оном из рада [1] се показао као много бољи где су његови резултати више него дупло бољи од сваке од засебних репрезентација што нам говори да те репрезентације правилно покривају дисјунктне случајеве. Али са погођених само 48% инстанци и он је далеко од жељеног. Мислим да би мало бољи резултати могли да се добију померањем тежина при гласању, али за бољи напредак мислим да би се у гласање морао додати још један класификатор истриниран над другачијом репрезентацијом текста. Идеја за тај последњи класификатор је да буде урађен над репрезентацијом текста у виду н-грама карактера у који улазе само знаци интерпункције. Још и у прошлом раду сам увидео значај знакова интерпункције на резултате, и зато сматрам да би додатак такве репрезентације донео побољшање при гласању. Ако погледамо резултате мог пређашњег рада [2] можемо увидети да су овде добијени резултати знатно гори. Пошто сам овде користио исте репрезентацију и исти класификатор у случају одвојених репрезентација а резултати су и даље гори увиђам само два могућа разлога за овакав исход:

1. У овом раду тест сет је дупло већи и чини трећину уместо четвртине тренинг сета.

2. У прошлом раду је постојало много више исечака и делова текста у њима који им не припадају што је можда олакшавало класификацију. Ово је знатнији проблем јер компромитује валидност пређашњег рада.

Литература

[1] Lukas Muttenthaler, Gordon Lucas, Janek Amann: Authorship Attribution in Fan-Fictional Texts given variable length Character and Word N-Grams Notebook for PAN at CLEF 2019, University of Copenhagen

[2] Никола Радојчић, Препознавање аутора на основу исечка текста са акцентом репрезентације текста преко н-грама речи