Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Krcsmárik Robin

2019

Szegedi Tudományegyetem

Informatikai Intézet

Hibás osztály automatikus keresése hibajelentés

(bug report) alapján

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Krcsmárik Robin |  | Dr. Vidács László |  |
|  | informatika szakos hallgató |  | tudományos munkatárs |  |

Szeged

2019

## Feladatkiírás

Feladat a hibajelentés alapján (pl. https://bugs.webkit.org/show\_bug.cgi?id=122565) a forráskódban beazonosítani azon osztályokat, melyek nagy valószínűséggel hibásak, vagyis a hiba javításához őket kell módosítani. Megoldásként szolgálhat a hibajelentésben szereplő szavak és az osztályokban szereplő azonosító nevek közötti egyezések vizsgálata.

## Tartalmi összefoglaló

* A téma megnevezése:

Hibás osztály automatikus keresése hibajelentés(bug report) alapján

* A megadott feladat megfogalmazása:

A feladat egy olyan alkalmazás elkészítése, mely a megadott hibabejelentések alapján javaslatot tesz a legvalószínűbb hibás, azaz javításra szoruló osztályokra. Szétbontva áll egy szövegbányászat felhasználásával megközelíthető osztályozási problémára megoldást nyújtó eljárás felhasználásából, illetve egy azt alkalmazó program megvalósításából.

* A megoldási mód:

Az általam választott project melyen a feladatban megadott program dolgozik a bugzilla hibakezelő rendszerben kezelt Mozilla Gecko nyílt forráskódú verziókezelt projecthez kapcsolódó bejelentések részhalmaza (https://bugzilla.mozilla.org/). Ezek a részek java programnyelven készültek. A projekt forráskódja a https://github.com/mozilla/gecko-dev.git címen elérhető. Első körben az adatok gyűjtéséhez a mozilla project repository-ából a program kigyűjti a bejelentések alapján készített commitokat és a committal módosított fájl listát. Ezt követően a kigyűjtött bug azonosítók alapján a program lekérdezi a hibák adatait és leírásukat a bugzilláról. Következő lépésként szövegbányászati módszereket alkalmazva először Vektortérmodellt állít elő a program, melyen koszinusztávolságot és a dolgozatban, a későbbiekben részletesebben tárgyalt értékeket számol. Végül a megkapott értékek súlyozásával egy rangsor áll elő a szupportvektor-gép rangsoroló osztályozóként való alkalmazása során. Az osztályozó tanulása és tesztelése során a k-szoros keresztvalidációt alkalmazza a program. Eredményként egy lehetségesen javítandó top-k rangsorolt állomány listát ad.

* Alkalmazott eszközök, módszerek:

A program fejlesztése Eclipse 2018-09 segítségével Windows 10 operációs rendszeren készült, java 8 programnyelven. A program az adatbázis kezeléséhez a jdbc sqlite api-t, a git repository kezeléséhez a JGit api-t, a vektortérmodell előállításhoz az Apache OpenNLP api-t és az SVM rangsoroláshoz a C nyelven íródott SVM Rank programot használja. A program az előzőeken felül módszerként alkalmazza a k-szoros keresztvalidációt a szupportvektor-gép tanulásához és teszteléséhez

* Elért eredmények:

A feladat megvalósításával sikerült egy olyan programot létrehozni, mely képes meghatározni a javítandó osztályokat azonos területen más kutatási fejlesztés során elért eredményeivel azonos pontossággal.

* Kulcsszavak:

Eclipse, Java 8, Mozilla Gecko, Bugzilla, Ranking SVM, Apache OpenNLP, JGit

## Tartalomjegyzék

[Feladatkiírás 4](#_Toc6929869)

[Tartalmi összefoglaló 5](#_Toc6929870)

[Tartalomjegyzék 6](#_Toc6929871)

[BEVEZETÉS 7](#_Toc6929872)

[1. A szövegbányászat, mint módszer 8](#_Toc6929873)

[1.1. Előfeldolgozás és modellalkotás 8](#_Toc6929874)

[1.1.1 Szövegbányászat feladata 8](#_Toc6929875)

[1.1.2 Vektortérmodellalkotás 8](#_Toc6929876)

[1.1.3 Modell súlyozása 9](#_Toc6929877)

[1.1.4 Dokumentum feldolgozása vektortérmodellé 10](#_Toc6929878)

[1.2. Osztályozás 10](#_Toc6929879)

[1.3. A relevancia mérése 11](#_Toc6929880)

[2. Hibakeresés végrehajtása 12](#_Toc6929881)

[2.1. Adatgyűjtés 12](#_Toc6929882)

[2.2. Az atommag 12](#_Toc6929883)

[2.2.1. Az atommag szerkezete 12](#_Toc6929884)

[2.2.2. 12](#_Toc6929885)

[2.2.3. 12](#_Toc6929886)

[3. Kiértékelés 13](#_Toc6929887)

[3.1. 13](#_Toc6929888)

[3.2. 13](#_Toc6929889)

[3.3. 13](#_Toc6929890)

[4. ATOMMAG KÉSZÍTÉSE BARACKMAGBÓL 14](#_Toc6929891)

[4.1. 14](#_Toc6929892)

[4.2. 14](#_Toc6929893)

[4.3. 14](#_Toc6929894)

[Irodalomjegyzék 15](#_Toc6929895)

[Nyilatkozat 16](#_Toc6929896)

[Köszönetnyilvánítás 17](#_Toc6929897)

# BEVEZETÉS

Egy szoftver fejlesztési projekt sikeréhez létfontosságú a szoftver minősége. A színvonal javítása érdekében a fejlesztés során számos szoftverminőség-biztosítási tevékenységet (, mint például teszteléseket, statikus teszteléseket, ellenőrzéseket stb.) hajtanak végre a projekt résztvevői. Viszont az elkészült rendszerek még így is számos hibákat tartalmaznak a valóságban. Egy nagy és fejlődő szoftverrendszer úgymint a Mozilla Gecko is hosszú időn keresztül nagyszámú hibabejelentést kaphat. Az idei év március hónapjában például a Mozilla Gecko projekt több mint 3000 javításon vagy módosításon esett át. Egy átlagos hibajavítás során a hibabejelentés megtörténte, megerősítése is továbbítása után a fejlesztőknek meg kell találnia azokat a forráskódfájlokat, amelyeket javítani vagy módosítani szükséges a hiba javításának érdekében. Ezen állományok kézi felkutatása egy igen időigényes, ezáltal költséges feladat, különösen egy nagy projekt esetében ahol a kód több ezer állományból áll.

Éppen ezért az utóbbi években számos kutatás indult információ visszakeresésen alapuló technikát alkalmazó módszerek kifejlesztésére irányulóan annak érdekében, hogy a hibabejelentéshez tartozó módosítandó forrásfájlok automatikusan kerüljenek meghatározásra. Ennek alapján ezek a kutatások a hibabejelentést lekérdezésként kezelik, melynek eredménye a javítandó forrásfájlok javasolt listája. A lista alapján javítható a bejelentett hiba, így kikerülve az időigényes forráskód keresést.

Ez a fajta hibakeresési módszer a hibabejelentések és a forráskódok szöveghasonlóságára épülve az információ visszakeresésen (IR) alapszik, így az nem igényel program végrehajtási információkezelést (pl. stack trace kezelést stb.).

Az előzőekben említett kutatások eredménye például a BugScout [1], mely a hibajelentések és a programkód teljes szövege között keres hasonlóságot szövegbányászati módszerrel.

Egy másik kutatás eredménye a BugLocator[2], mely a rendelkezésre álló hibabejelentések alapján bejelentésenként a vektortér-modell előállítása segítségével rangsorolja a forráskódfájlokat relevancia szerint, használva a Látens Dirichlet allokációt (LSI) és a Látens szemantikus indexelést (LSI).

A dolgozatomban egy olyan korábbi Learning to Rank Relevant Files for Bug Reports using Domain Knowledge [3] című kutatásban említett módszert implementáltam, mely hasonlóan az előzőekben említettekhez információ visszakeresésen alapszik. A hibabejelentésekhez a forráskódok relevanciáját szupportvektor-gép osztályozó rangsorolja.

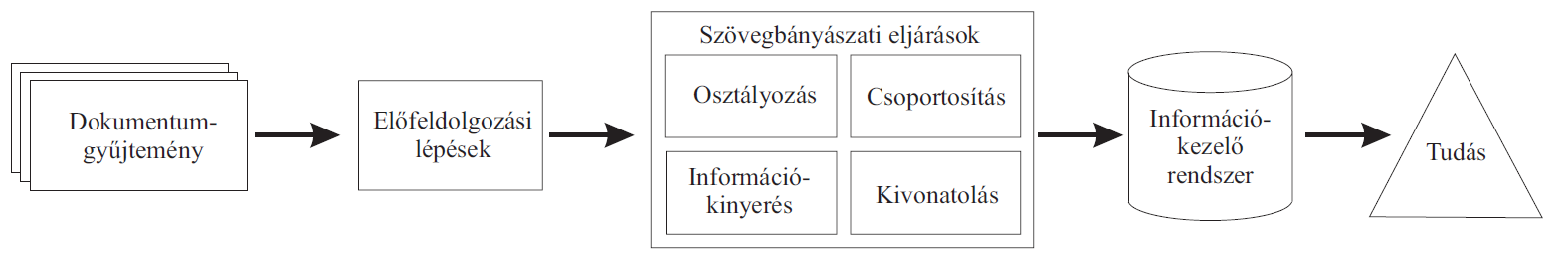
# 1. A szövegbányászat, mint módszer

## Előfeldolgozás és modellalkotás

### Szövegbányászat feladata

A szövegbányászat az emberek közötti szóbeli vagy írásbeli kommunikáció céljára alakult ki nem a számítógépes feldolgozás szerint. Az emberek könnyedén felismerik és alkalmazzák a nyelvi mintákat. Nem okoznak nekik gondot a helyesírási variációk kezelése, a kontextus felismerése vagy a stilisztikai jelleg azonosítása. Viszont nincs meg bennünk a számítógépeknek az a képessége, hogy a szöveget nagy mennyiségben, vagy nagy sebességgel

dolgozzuk fel. A szövegbányászat feladta tehát az emberi nyelvi tudás ötvözése a számítógép nagy feldolgozási képességével.

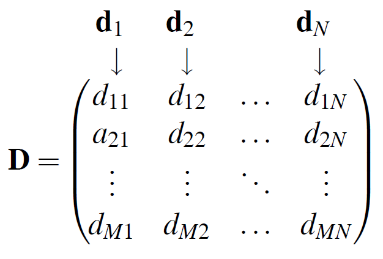


1.1.1.1 Ábra: A szövegbányászat modellje [4]

### Vektortérmodellalkotás

A szövegbányászati feladatoknak két altípusa, van a keresés és a rendszerezés. Az előbbi esetben olyan dokumentumokat keresünk, amelyekben egy adott keresőkifejezés fordul elő, utóbbiban pedig dokumentumokat hasonlítunk össze, majd ennek alapján kapcsoljuk őket valamely kategóriarendszer elemeihez. A dolgozatomnál a rendszerezés típusú szövegbányászati feladatot kell végrehajtani, ami más megközelítést igényel a kereséshez képest.

Ahhoz, hogy a dokumentumgyűjtemény elemein valamilyen rendszerezést hajtsunk végre, szükségünk van egy olyan modellre melynek segítségével a dokumentumok hasonlóságát, azaz a távolságát mérni tudjuk. Alapgondolatként azok a dokumentumok hasonlítanak egymásra, amelyeknek a szókészlete átfedik egymást és a hasonlóság mértéke az átfedéssel arányos. Ezt használja az információ-visszakeresésben használt vektortérmodell.



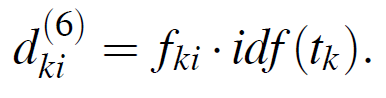
1.1.2.1 Ábra: Vektortérmodell Szó-Dokumentum mátrixa [5]

A vektortérmodell egy vektortérben ábrázolja a vizsgálandó illetve összehasonlítandó dokumentumokat. A modell egyik dimenziója a dokumentumgyűjteményben lévő dokumentumok(N), míg a másik dimenziója dokumentumgyűjteményben előforduló egyedi szavak(M). A vektortérmodell sajátossága, hogy a szavak szövegen belüli pozíciója és sorrendje elvész. Ezt a megközelítést szózsákmodellnek is nevezi a szakirodalom.

### Modell súlyozása

A vektortérmodell azaz a kulcsszó-dokumentum mátrix elemeit (, ahol a sorok a kulcsszavak és az oszlopok a dokumentumok) többféleképpen is lehet súlyozni. Lehet többek között például bináris súlyozással, miszerint ha a szó előfordul a dokumentumban, akkor 1 az érték, egyéb esetben 0. Előfordulás alapú súlyozásnál már nemcsak az számít, hogy a szót tartalmazza a dokumentum, hanem az is hogy hányszor. Ez a darabszám lesz az értéke az elemnek. Logaritmikus súlyozásnál az előfordulás alapú súlynak vesszük a logaritmusát és hozzáadunk egyet.

Az eddig említett súlyok nem vették figyelembe a dokumentum hosszát. A nyelvi kifejezés jobb közelítése érdekében helyi és átfogó súlyok bevezetése szükséges, ahol a helyi súly az aktuális dokumentumra tipikusan az előfordulási számra, míg a globális a kulcsszóra vonatkozik. Ennek a súlyozási sémának a szógyakoriság és inverz dokumentumgyakoriság index (TF-IDF index) a neve.



1.1.3.1 Ábra: TF-IDF súlyozás képlete [5]

Ahol az fki a tk szó di dokumentumbeli gyakorisága (azaz a szó előfordulási számát osztjuk a dokumentumbeli szavak számával).

Az idf(tk) pedig a dokumentumgyakoriság inverze, ami egyenlő a log(N/nk), ahol nk azon dokumentumok száma ahol tk előfordul, N pedig a dokumentumok száma.

Az így kialakult TF-IDF súlyozás értéke tehát magas lesz azon szavak esetében, amelyek az adott dokumentumban gyakran fordulnak elő, míg a teljes korpuszban ritkán (így nagy a megkülönböztető erejük). Alacsonyabb lesz a súlyozás értéke viszont azon szavak esetében, amelyek a dokumentumban ritkábban, vagy a korpuszban gyakrabban fordulnak elő. Harmadik esetben pedig csekély akár zérus értékű lesz azon szavak esetében, amelyek szinte a korpusz összes dokumentumában előfordulnak.

### Dokumentum feldolgozása vektortérmodellé

Ahhoz, hogy egy szöveges dokumentumot vektortérmodellben ábrázolni tudjuk a súlyozási séma kiválasztásán kívül további előfeldolgozási lépéseket kell megtenni.

Először a szöveges dokumentumokat fel kell bontani további egységekre, jellemzően szavakra. Ezt a szakirodalom tokenizálásnak hívja. Tokennek hívjuk egy karaktersorozat konkrét előfordulását, míg típusnak hívjuk az azonos karaktersorozatot tartalmazó tokenek osztályát, azaz a típusok összessége alapján (esetleges további szükséges feldolgozási lépések közbeiktatásával) áll elő a szótár.

Következő lépés az úgynevezett stopszó szűrés, ahol a gyakran előforduló, tartalmi információt egyáltalán nem tartalmazó, megkülönböztető képesség nélküli stopszavak eldobásra kerülnek.

Ezt követően a szavaknak át kell esniük a lemmatizáláson és a szótövezésen. A legtöbb nyelvben a szavak előfordulnak toldalékolt vagy módosított alakban is. A vektortérmodell kialakításánál szükséges az azonos szavaknak különböző szóalakú előfordulásait közös kanonikus alakra hozni. A kanonikus alak meghatározásához a szakirodalom két megközelítést használ. Lemmatizálásnak nevezi a szó lemmájának (normalizált, vagy szótári alakjának) előállítását, illetve meghatározását. Szótövezésnek hívja azt az eljárást, amikor az adott szó szótövének meghatározása a cél. A két eljárás között az a különbség, hogy míg a nyelvészeti motivációjú lemmatizálás mindig értelmes szóalakot állít elő, addig a szótövezés során jellemzően a szó csonkolása történik.

## Osztályozás

### Osztályozás feladata, típusai

A szövegbányászat egyik alapfeladata a szövegek osztályozása. Az osztályozásnak két eltérő megközelítése is létezik. Ez a kettő a klaszterezés és a kategorizálás. A klaszterezés automatikus módszer, ahol az algoritmus maga alakítja ki a kategóriákat. A dolgozat szempontjából az utóbbi megközelítés, azaz a kategorizálás az érdekes.

Formálisan: Φ : D 🡪 2C osztályozáskor egy osztályozófüggvény megalkotása a cél, amely a D dokumentumtér elemeihez a C = {c1, . . . ,c|C|} kategóriarendszerből vett kategóriák halmazát rendeli. A Φ függvényt röviden osztályozónak nevezik. Az osztályozó létrehozásakor a cél az ismeretlen Φ˜ : D → 2C célfüggvény minél pontosabb közelítése, azaz Φ és Φ˜ eltérésének minimalizálása.

Az osztályozás a kategória címke számtól függően lehet egy vagy több címkés, illetve az osztályozás használata szerint kategória vagy dokumentum vezérelt.

A dokumentumvezérelt osztályozásnál adott d ∈ D esetén keressük az összes d-hez tartozó kategóriát. A kategóriavezérelt osztályozásnál adott c ∈ C kategória esetén keressük a c-be tartozó összes dokumentumot.

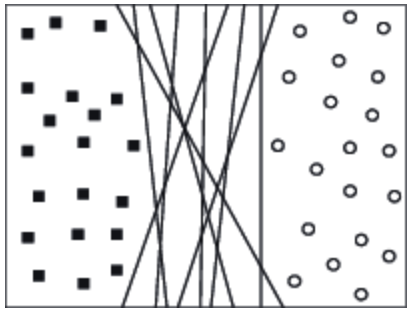
Az eredmény típusa szerint is kétféle osztályozást, a kiválasztó és rangsoroló osztályozást különböztetjük meg. A rangsoroló osztályozásnál a Φ : D x C 🡪 [0,1] függvény nem csak 0 vagy 1 értéket vehet fel, hanem ez az érték egy valós szám lesz. Így a dokumentumvezérelt esetben egy adott d ∈ D dokumentumhoz rendelt kategóriák relevanciáját a C elemeihez rendelt érték alapján lehet rangsorba állítani.

### Az SVM osztályozó algoritmus

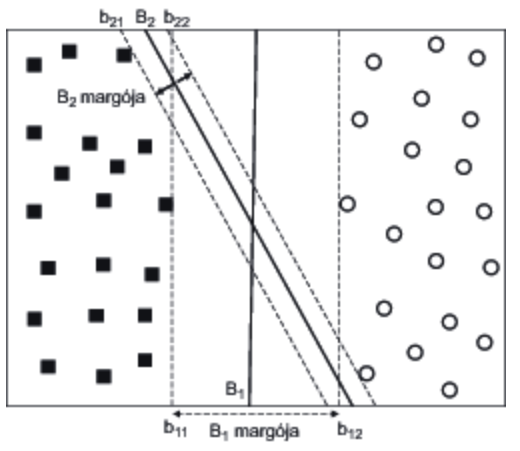
A szupportvektor-gép (Support Vector Machine – SVM) osztályozó a szövegbányászat terén az egyik leghatékonyabb gépi tanulási módszeren alapuló módszer. Az SVM alapja a lineáris

osztályozók csoportjába tartozik és bináris osztályozási problémák megoldására alkalmas.

Maga az algoritmus egy olyan döntési hipersíkot határoz meg, amely nem csak elválasztja a negatív tanítóadatokat a pozitív tanítóadatoktól, hanem maximalizálja a mintáktól való távolságát is a hipersíknak (maximális margójú hipersíkot állít elő).



* + - 1. Ábra: Lehetséges döntési határok lineárisan szeparálható adatok esetén [6]



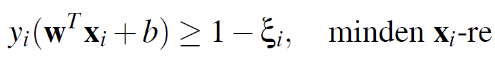
* + - 1. Ábra: Döntési határ margója [6]

A tanítóadatokat szupportvektoroknak nevezzük. A hipersík meghatározásában a tanítóadatok közül csak a szupportvektorok játszanak szerepet.

A nagy margóval rendelkező döntési határoknak általában jobb az általánosítási hibájuk, mint a kis margóval rendelkezőknek. Viszont ha a margó kicsi, akkor a döntési határ bármilyen kis zavarának elég jelentős hatása lehet az osztályozásra. A kis margóval rendelkező döntési határokat létrehozó osztályozók ezért hajlamosabbak a modell túlillesztésre és a korábban nem látott eseteken gyakran rosszul általánosítanak.

Nagy adathalmazok esetén a szeparabilitási feltétel gyakran nem teljesül (nem szeparábilis eset), illetve lineárisan szeparálható halmazok esetén is akadnak olyan kilógó, izolált vagy zajos pontok, amelyeket jobb figyelmen kívül hagyni az osztályozó létrehozásánál.

Ebben az esetben gyengítő változók bevezetése válik szükségessé és a hipersík egyenlet kis módosítással érvényben marad.



* + - 1. Ábra: Hipersík egyenlete gyengítő változóval [7]

A feladat megoldásánál a tartalék szélessége és a hibásan osztályozott, és a biztonsági sávba benyúló pontok száma közötti optimalizálást hajtunk végre. Fontos megemlíteni az SVM osztályozóknál megadható C regularizációs faktort, amellyel az adatok fontossága szabályozható. A kicsi jobban tolerálja a túllógó pontokat, mint a nagy érték.

## A relevancia mérése

# Hibakeresés végrehajtása

## Adatgyűjtés

## Az atommag

### Az atommag szerkezete

### 2.2.2.

### 2.2.3.

# 3. Kiértékelés

## 3.1.

## 3.2.

## 3.3.

# 4. ATOMMAG KÉSZÍTÉSE BARACKMAGBÓL

## 4.1.

## 4.2.

## 4.3.

## Irodalomjegyzék

[1] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, Tien N. Nguyen - A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report

[2] Jian Zhou, Hongyu Zhang, and David Lo - Where Should the Bugs Be Fixed? More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports

[3] Xin Ye, Razvan Bunescu, and Chang Liu - Learning to Rank Relevant Files for Bug Reports using Domain Knowledge

[4] Tikk Domonkos – Szövegbányászat, a szövegbányászat feladata, TypoTex, 2007

[5] Tikk Domonkos – Szövegbányászat, dokumentum reprezentálása vektortérmodellben, TypoTex, 2007

[6] <https://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_adatbanyaszat/ch05s05.html>

[7] Tikk Domonkos – Szövegbányászat, Osztályozás, TypoTex, 2007

## Nyilatkozat

Alulírott Krcsmárik Robin programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, Programtervező Informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2019. április 23.

Aláírás

## Köszönetnyilvánítás