Járművekről szóló vélemények automatikus érzelemelemző rendszere

(Angol cím: Towards automated vehicle profiling)

# Bevezető

A „Big data” korszak úton van. A “Big data” óriási adatmennyiséget jelent ami a világ minden tájáról, napról napra, óráról órára előállitodik intelligens hálozatok, magánszektor és egyéni felhasználok által. Különböző szakterületű tudósok, mint számítógép-tudósok, fizikusok, matematikusok, politológusok, bio-informatikusok, szociológusok, sürgetik a hozzáférést ehhez az óriási adathoz amiket az emberek, dolgok és a köztük létező kölcsönhatások hoznak létre. [1]

1. A „Big data” négy tulajdonsága a 4V :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Mennyiség**  **(Volume)** |  | **Sebesség (Velocity)** |  | **Változatosság (Variety)** |  | **Igazmondás (Veracity)** |
|  |  |  |  |  |  |  |
| **Adat**  **nyugalomban** |  | **Adat**  **mozgásban** |  | **Adat változatosságban** |  | **Adat**  **kételyben** |
| Terrabájt és exabájt nagyságú feldolgozásra váró létező adat |  | Folyamló adat, milimásodpercekttől másodpercekig tartó válaszidő |  | Strúkturált, strúkturálatlan, szöveg, multimédia |  | Bizonytalanság ami az adat inkonszisztenciájából és teljesség hiányából, kettősértelműségéből, lappangásából, tévedésekből, minták megközelítéséből származik |

A mennyiség a másodpercenként előállított hatalmas adatözönre vonatkozik. A sebesség fontos mivel az adatok nem halmazokban jönnek hanem folyamatosan áramlanak, egyre nagyobb sebességgel és ezeknek az adatoknak a valós időben történő feldolgozása egy igazi kihívás. Az adatok változatossága megköveteli az egyes adatok strukturálását és egymással összefüggésbe hozását forrásra tekintet nélkül. Az adatok igazmondása szintén egy fontos jellemzője a „Big data” – nak, ez nagyon változatos az adatforrások között . A cél a kontrollálatlan adatfolyamok formázása az értékes információk kinyeréséhez, ami hozzásegíthet üzleti döntések meghozatalához és hosszú távú versenyelőnyök megszerzéséhez.

A „Big data” felmérésével a menedzserek több információt megtudhatnak az üzletjükről és ez az ismeret segíti a jobb döntés-hozásban és jobb teljesítmény elérésében. A „Big data” használata megengedi a menedzsereknek hogy a döntéseket bizonyítékokra alapozzák az ösztöneik helyett. Azok a cégek akik felölelik a bizonyítékokra alapozó döntéshozást olyan tudóst kell alkalmazzon aki mintákat tud találni az adatokban és ezeket lefordítja hasznos üzleti információvá. A „Big data” fogalmat sokszor asszóciálják a következő két ötlettel: adat tárolás és adat elemzés. [2]

A közösségi média egy olyan médiaeszköz, ahol az üzenet közösségi interakciókon keresztül szórodik szét. Az internetet és az online megjelenésítést használja fel azzal a céllal, hogy a média monológokat átalakítsa dialógusokká. A közösségi médiának fontos szerep van a marketting szempontjából. A közösségi médiának két kapcsolatban lévő előremozdító szerpe van a piactéren. Először is megengedi a cégeknek az ügyfeleikkel való beszélgetést és másodszor megengedi az ügyfeleknek az egymás közti beszélgetést. A közösségi média felerősítette a felhasználó – felhasználó közötti kommúnikációt, hisz lehetőséget kínál arra, hogy egy felhasználó több száz vagy akár ezer más felhasználóval kommunikáljon a piactéren, gyorsan és kis erőfeszítéssel. [3]

Érzelem elemzés

Mindig kiváncsiak voltak az emberek a következő információra “mit gondol a másik személy” valamiről vagy valakiről, majd erre az információra alapozva döntéseket hoztak. [4] Ezek a vélemények amiket a személyek megfogalamznak érzésekből fakadnak. Az érzések szubjektív benyomások nem pedig tények. Az érzések hozzáállásban, érzelmekben és véleményekben nyilvánulhatnak ki. Az érzelem elemzés felhasználva szövegelemzőket, statisztikákat vagy géptanúlási technikákat kibányász, meghatároz vagy másképp fogalmazva jellemzi az érzelmi tartalmát egy adott szövegnek. Bizonyos esetekben úgy is hívatkoznak az érzelem elemzésre mint vélemény kutatás. Az érzelem elemzés a következő kérdéseket fogalmazhatja meg:

* Pozitív vagy negatív a termékről szoló véleményezés?
* Ez a vásárló emailje elégedettséget vagy elégedetlenséget tükröz?
* Hogy változott a bloggolók hozzáállása az elnökhöz a választások óta?

Az érzelem elemzés alkalmazható az üzleti világban. Választ adhat olyan kérdésekre mint: Miért a vásárlok nem a mi termékünket veszik meg? Meg szeretnénk tudni a szubjektiv adatokat mint például: “a kinézete csúf”, “a fékje rossz minőségű” “a motor hangja szuper” “az ülések kényelmesek”.

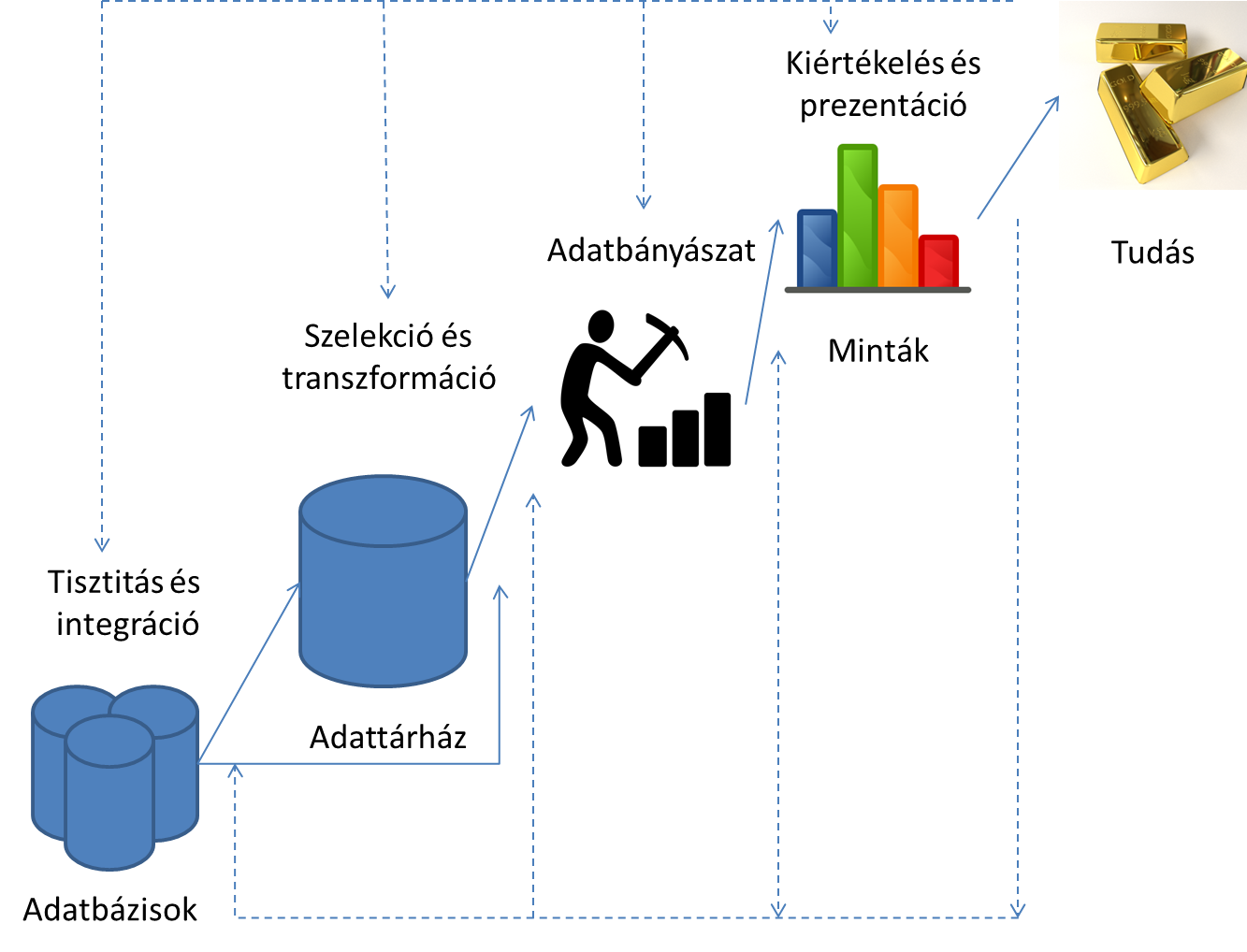
Mivel nagyon nehéz lenne a vásárlókat véleményeztetni olyan termékekről amit meg sem vásároltak, sokkal hatékonyabb a weben, szociális hálókon vagy a közösségi médiákban keresni véleményeket a saját termékünkről vagy akár a konkurenseink termékéről.

Az általam készített aplikáció a Twitter – ismeretségi hálózat csevegéseiből épít fel egy saját adatforrást. A csevegések szövegjét mondattani elemzéssel feldolgozza, megkeresi a kulcsszó – tulajdonság kapcsolatokat és az így felépített mondatrészre érzelmi kiértékelést végez. Az aplikáció segítségével a felhasználó értékes információt kap az ügyfelek érzelmeiről az adott termék bizonyos tulajdonságairól, íly módon megtudhatja melyek azok a tulajdonságok amivel az ügyfelek nem elégedettek és fejlesztést igényelnek, valamint azokat is amikkel elégedettek és bizonyára a termék megvásárlását, használatát indokolják. Ugyanakkor ez az elemzés elvégezhető konkurrens cégek termékeire is és összehasonlíthatjuk a sajátunkkal, információt kapva arról, hogy miben jobb vagy rosszabb a termék az ügyfelek véleményei alapján. Ezek az információk szolgálhatnak alapul a menedzsereknek a döntéshozásban, megmutatják melyek azok a területek ahol fejlesztés, befektetés szükséges.

# Bibliográfiai tanulmány

## Adatgyűjtés

„Az információ korában élünk” ezt tartja a népszerű mondás napjainkban, pedig inkább azt mondhatnánk, hogy az adat korában élünk. Terrabájt vagy petabájt nagyságú adatok özönlenek a számítógépes hálózatokon, a World Wide Web-en, és különböző adathordozó eszközökön naponta üzleti, társadalmi, tudományos és műszaki tudományok, orvostudomány, és a mindennapi életünk majdnem minden területéről. A közösségek és közösségi média adatinak a fontossága egyre növekedik és változatos formájú adatokat hoz létre mint: digitális képek és videók, blogok, Web közösségek, és különböző típusú szociális hálózatokat. Erőteljes, sokoldalú és flexibilis eszközökre van szükség amelyek automatizáltan kiemeljenek értékes információkat ezekből a hatamas adattömegekből és átalakitsák ezeket az adatokat szervezett tudássá. Az adatbányászat átalakít egy adattömeget tudássá. Sokan úgy tartják az adatbányászat szinonimája a “knowledge discovery from data” , ismeretek felfedezése adatokból, kifejezésnek. Mások nézete szerint az adatbányászat egy lépés az ismeret felfedezésének a folyamatában.



1. Ábra Adatbányászat folyamata

Az adatbányászat az a folyamat amely nagy terjedelmű adatokban érdekes adatmintákat és tudást fedez fel. Az adatforrások magukba foglalnak adatbázisokat, adattárházakat, a Web-et, más információs tárházakat, vagy állandoan folyamló adatokat egy dinamikus rendszerbe.

## Szöveg elemzés

A természetes nyelv feldolgozásának fő célja természetes nyelvet megértő és feldolgozó aplikációk készítése. Természetes nyelv alatt emberi lények által beszélt vagy írott nyelvet értünk, ellentétben a mesterséges, számítógépi, számtani vagy logikai nyelvvel, például.

A természetes nyelv feldolgozásának főbb szerepeként megemlítem a szófaji annotációt (POS part-of-speech), a nevezett-entitások felismerését és a szintaktikai vagy mondattani kapcsolatok annotációját. [5]

Itt említeném meg a korpuszannotációt, amely azokat az információkat és jeleket foglalja magába, amelyek az elemzett szövegben nincsenek benne, viszont az úgynevezett korpusz készítésekor vagy feldolgozásakor belekerülnek a szövegbe. A szófaji címkézés során a szövegbe minden egyes szó mellett meg fog jelenni a szófaji megjelölés is. [6] A szintaktikai annotációt egyes generatív nyelvészeti elemzők, mint például a Stanford Parser, az ismert ágrajzzal, amit angolul tree-nek neveznek, ábrázolnak.

Felhasználva a Stanford Parser online weboldalas felületét a következő mondatra:

“BMW has a powerful engine, but VW has the best brakes.” a következő eredményeket kapjuk:

* szófaji annotáció

BMW/NNP has/VBZ a/DT powerful/JJ engine/NN ,/, but/CC VW/NNP has/VBZ the/DT best/JJS brakes/NNS ./.

Ezen a példán láthatjuk a szavak mellet a megfelelő szófajt: „BMW” főnév tulajdonnév, „has” ige, „a” határozószó, „powerful” melléknév, „but” mellérendelő kötőszó, „VW” főnév tulajdonnév, „has” ige, „the” meghatározó szó, „best” felsőfokú melléknév, „brakes” főnév többes szám.

* Szintaktikai kapcsolatok annotációja

(ROOT

(S

(S

(NP (NNP BMW))

(VP (VBZ has)

(NP (DT a) (JJ powerful) (NN engine))))

(, ,)

(CC but)

(S

(NP (NNP VW))

(VP (VBZ has)

(NP (DT the) (JJS best) (NNS brakes))))

(. .)))

A nevezett entitások felismerése arra vonatkozik, hogy a szövegben felismer és osztályoz szövegelemeket előre meghatározott kategóriákba, mint például személynevek, szervezetek, pénznemek, helységek. Az általam bemutatott példamondat esetében a BMW és VW szavakat osztályozná a cégek vagy autómárkák kategoriájába.

Egy természetes nyelv szövegelemző egy olyan aplikáció amely előállitja a mondatok nyelvtani struktúráját, mely szó-csoportok alkotnak együtt egy frázist, és mely szavak az alany vagy tárgy részei egy igének. [7]

## Érzelem elemzés

Az érzelem analízis, más néven véleménybányászat, az a tudományterület mely az emberek termékekről, szolgáltatásokról, szervezetekről, egyénekről, eseményekről, témakörökről alkotott véleményét, érzelmét és attitűdjét elemzi [8]. Ez a tudományterület több névén is ismert, de minden név kissé eltérő feladatokat feltételez: vélemény kitermelés, érzelem bányászat, szubjektivitáselemzés, emóció analízis, ismertetéselemzés. Az érzelem analízis kifejezést 2003-ban használta először Nasukawa és Yi, habár a kutatások már a 2000-től folytak ezen a területen. Mielőtt döntenénk egy termék megvásárlásában, véleményeket gyűjtünk róla, megkérdezzük a barátainkat, online üzletek oldalain elolvassuk a véleményeket, fórumokat böngészünk, vagyis véleménybányászatot végzünk. Egy vállalat szemszögéből is fontos, hogy mit gondolnak a termékeinek a fogyasztói, felhasználói a gyártott termékekről. Ezért nem csak egyéni, hanem üzleti szempontból is fontossá vált, ennek a tudományterületnek a kutatása, fejlesztése. Így ez a tudományterület lassan fel fogja váltani a gyártók, üzletláncok által terjesztett felmérő-, kiértékelő lapokat.

Az érzelemelemzést szintekre lehet bontani. Az első szint a dokumentum szintű érzelemelemzés. Ennek során egy egész dokumentum alapján dönti el a rendszer, hogy összességében a vélemény pozitív vagy negatív. A második szint a mondat szintű érzelemelemzés. Itt a rendszer azt dönti el, hogy a mondatban pozitív, negatív vagy semleges vélemény fogalmazódik meg. A harmadik az entitás és aspektus szintű elemzés. Azon az ötleten alapul, hogy egy vélemény egy érzelemből (legyen az pozitív vagy negatív) és egy célpontból áll. A vélemény a célpont nélkül fölösleges. A következő mondat az aspektus szintű érzelem analízis fontosságát szemlélteti: „Annak ellenére, hogy a székek nem kényelmesek szeretem az autómat”. Ha entitás és aspektus szinten elemezzük ki, akkor az autóról pozitív a véleményem, a benne levő székekről viszont negatív.

Az érzelem-elemző rendszerek többsége úgy működik, hogy egy előre definiált érzelemlexikont használnak. Az érzelemlexikon olyan szavakból áll, melyekhez valamilyen érzelem kapcsolódik. A „jó”, „szép”, „csodás”, „elbűvölő”, „fantasztikus” mind-mind pozitív érzelmeket sugároznak. A „rossz”, „gyenge”, „rémes”, „elviselhetetlen” szavakhoz viszont negatív érzelmek fűződnek. A probléma ott van, hogy csak érzelemlexikont használva nem kapunk mindig pontos eredményeket. Előfordulhat, hogy egy pozitív vagy negatív érzelmet sugároz, a mondat felépítésétől függően. Vegyük példának a „szívás” szót. A „Ez a telefon egy szívás.” és „Ennek a porszívónak jó a szívása.” mondatok jól szemléltetik a fontosságát a mondat felépítésének. Egy másik eset, amikor nem elégséges az érzelemlexikon használata, amikor a mondatban jelen van az érzelmet sugárzó szó, viszont a jelenléte semleges a véleményezés szempontjából: „Ha találok egy jó TV-t az üzletben, megvásárolom.” A harmadik eset, amikor az érzelemlexikont használva téved egy rendszer a szarkazmus, gúnyolódás esete: „Micsoda szuper telefon! Négy óra után lemerült az akkuja.”

# Elméleti megalapozás

Ebben a fejezetben bemutatom a Stanford Parser mondattani elemzését, kitérek a névelemek és tulajdonságok keresésére, a tulajdonság és névelem közti relációk felismerésére majd végül az érzelem meghatározására.

## Stanford Parser, Mondattani elemzés

A Stanford szövegelemző biztosít egy Nemzetközi Függőségi és Stanford Függőségi kimenetet, valamint mondat szerkezetű fákat. Ezek az úgynevezett függőségek úgy is ismertek mint nyelvtani kapcsolatok. A Stanford függőségi reprezentációja úgy volt megtervezve, hogy egy egyszerű leírást nyújtson a mondatban szereplő nyelvtani kapcsolatokról, könnyen megérthető és hatákonyan használható legyen olyan személyek által akik szövegek közti kapcsolatokat akarnak kinyerni anélkül, hogy magass nyelvtani szaktudással rendelkeznének. A jelenlegi Stanford szövegelemző függőségi definiciói megközelitőleg 50 nyelvtani kapcsolatot tartalmaznak. A függőségek mind bináris kapcsolatok: a nyelvtani kapcsolat két részből áll, a vezető (írányító, fej) és az alárendelt (függő). A függőségi definiciók felhasználják a Penn – Treebank part-of-speech és mondattani címkéket.

Annak érdekében, hogy egy adott szövegben névelemek és tulajdonságok közötti kapcsolatokat megtaláljunk ismernünk kell a mondatban szereplő szavak közti kapcsolatokat és a szavak szófaját.

* Elemzési fák [7]

A függőségek ábrázolása a Stanford Parser használata esetében 5 stílusban lehetséges:

* alap
* összevont
* propagálodó összevont
* összevont fa szerkezetet megőrző
* nem-összevont

A külonböző függőség típusok bemutatására a következő példamondatot fogom használni:

“Bell, a company which is based in LA, makes and distributes computer products.”

Az alap függőségek a Penn Treebank által definiált szófaj és frázis címkézést használják és fa szerkezetet alkotnak. A mondtban mindegyik szó pontosan egy másik dologtól függ, vagy egy másik mondatban szereplő szótól vagy az úgynevezett ROOT-0 (gyökér) kiváló jelölőtől.

Az összevont függőségek ábrázolásában, az elöljárószók, kötőszók, valamint az információk a kapcsolodó hivatkozásokról összevonódnak, hogy a szavak közti közvetlen függőségeket mutassák. Az összevont függőségre megtekinthetjük a következő példát:

prep(based-7, in-8) és pobj(in-8, LA-9) helyett prep\_in(based-7, LA-9).

A propagálodó összevont függőségek esetben, azokban a mondatokban ahol létezik kötőszó, a függőség propagálodik a kötőszó által összekötött szavakra. A mi példamondatunk esetében a két ige közötti kötőszó miatt, az összevont függőséghez két új függőséget kéne létrehozzon. A két ige “makes” és “distribute” között levő kötöszó miatt, az alany és tárgy kapcsolat ami a kötőszó első elemére, a “makes”-re, vontakozik tovább kéne propagálodjon a kötőszó második elemére is, a “distribute” –re:

nsubj(makes-11, Bell-1) propagált függőség nsubj(distributes-13, Bell-1)

dobj(makes-11, products-15) dobj(distributes-13, products-15)

Mivel ez az ábrázolás egy kibővítése az összevont függőségnek,ezért nem garantált a fa szerkezetű ábrázolás.

Az összevont fa szerkezetet megörző ábrázolásban azok a függőségek amelyek nem biztosítják a fa szerkezetnek a megőrzését ki fognak maradni.

A nem-összevont ábrázolás tartalmazza az alap függőségi ábrázolást és az extra ábrázolásokat is, melyek elrontják a fa strukturát, viszont nem propagál és össze sem von semmiféle kapcsolatokat.

2. Stanford Parser függőségi ábrázolások

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Alap | Összevont | Propagálodó összevont | Összevont fa | Nem-összevont |
| nsubj(makes,Bell) | nsubj(makes,Bell) | nsubj(makes,Bell) | nsubj(makes,Bell) | nsubj(makes,Bell) |
|  |  | nsubj(distibutes,Bell) |  |  |
| det(company,a) | det(company,a) | det(company,a) | det(company,a) | det(company,a) |
| appos(Bell,company) | appos(Bell,company) | appos(Bell,company) | appos(Bell,company) | appos(Bell,company) |
|  |  |  |  | ref(company,which) |
| nsubjpass(based,wich) | nsubjpass(based,company) | nsubjpass(based,company) | nsubjpass(based,company) | nsubjpass(based,wich) |
| auxpass(based,is) | auxpass(based,is) | auxpass(based,is) | auxpass(based,is) | auxpass(based,is) |
| rcmod(company,based) | rcmod(company,based) | rcmod(company,based) | rcmod(company,based) | rcmod(company,based) |
| prep(based,in) |  |  |  | prep(based,in) |
|  | prep\_in(based,LA) | prep\_in(based,LA) | prep\_in(based,LA) |  |
| pobj(in,LA) |  |  |  | pobj(in,LA) |
| root(ROOT,makes) | root(ROOT,makes) | root(ROOT,makes) | root(ROOT,makes) | root(ROOT,makes) |
| cc(makes,and) |  |  |  | cc(makes,and) |
|  | conj\_and(makes,distributes) | conj\_and(makes,distributes) | conj\_and(makes,distributes) |  |
| conj(makes,distributes) |  |  |  | conj(makes,distributes) |
| nn(products,computer) | nn(products,computer) | nn(products,computer) | nn(products,computer) | nn(products,computer) |
| dobj(makes,products) | dobj(makes,products) | dobj(makes,products) | dobj(makes,products) | dobj(makes,products) |

* Szófajok: JJ (melléknév), VBZ (ige), stb.

A Stanford Parser által használt Penn Treebank definiciója szerint a következő szófaj-annotációk léteznek:

|  |  |  |  |
| --- | --- | --- | --- |
| Címke | Szófaj | Címke | Szófaj |
| CC | Mellérendelő kötőszó | PRP | Személyes névmás |
| CD | Tőszámnév | PRP$ | Birtokos névmás |
| DT | Meghatározó | RB | Határozószó |
| EX | Létező | RBR | Összehasonlító határozószó |
| FW | Idegen szó | RBS | Felsőfokú határozószó |
| IN | Elöljárószó | RP | Igekötő |
| JJ | Melléknév | SYM | Jelkép |
| JJR | Összehasonlító melléknév | TO |  |
| JJS | Felsőfokú melléknév | UH | Indulatszó |
| LS | Listatétel márker | VB | Ige, alap formában |
| MD | Modósítószó | VBD | Ige, múlt idő |
| NN | Főnév, egyes szám és tömeges | VBG, VBN,  VBP,VBZ | Ige, más időkben |
| NNS | Főnév, többes szám | WDT | Wh – határozószó(determiner) |
| NNP | Tulajdonnév, egyes szám | WP | Wh – névmás |
| NNPS | Tulajdonnév, többes szám | WP$ | Wh – birtokos névmás |
| PDT | Predeterminer | WRB | Wh – határozószó(adverb) |
| POS | Birtokossági végződés |  |  |

## Névelemek (kulcsszavak) keresése

A névelemek kerséséhez feltételezzük, hogy a névelemek a szófaji annotáció során főnév annotációval lesznek ellátva. Mivel a járművek esetében egy adott autó-márkának általában több modellje van, ha a névelem az autó-márka, akkor szükséges annak az ismerete, hogy melyik modellek tartoznak az adott autó-márkához. Az aplikáció a szinonimák bevezetésével oldja meg ezt a szükséget. Tehát például a Volkswagen autó-márka névelemhez hozzárendelhetünk több szinonimát mint Golf, Tiguan, Passat amik az adott autó-márka modelljei.

A felmerülő problémák közt megemlítem a névelem felismerését az egyes és többes szám esetében, valamint ha a névelem több szóból áll.

## Tulajdonságok keresése

A tulajdonságok esetében is azt feltételezzük, hogy a szófaji annotáció során ők is főnévi jelölést kapnak.

Tekintettel a szavak különböző alakjaikra, amelyek ragozás és képzők használata során jönnek létre, fontos ezeknek az előfordulásait közös kanonikus alakban összevonni. Ennek a kanonikus alaknak a meghatározása érdekében szócsonkolást alkalmazunk. Két megközelítést használ a szakirodalom:

* nyelvészetben lemmatizálás, melynek során értelmes szóalakot állít elő
* alkalmazásorientált számítogépes nyelvészeti szakirodalom (szó)tövezés, melynek során a szó szótövét határozzuk meg, ilyenkor a szó csonkolása történik és nem mindig kapunk értelmes szótári alakot.

Az alábbi táblázatban néhány példát tekinthetünk meg a két módszerről és a vissztérített eredményekről:

|  |  |  |  |
| --- | --- | --- | --- |
| Nyelv | Szavak | Lemmatizálás | Tövezés |
| Angol | works,worked,working | work | work |
| Angol | loves,loving, loved | love | lov |
| magyar | munkát, munkám | munka | munka |
| magyar | lovak, ló, lóvát | ló | lo , ló |

## Tulajdonság – névelem relációk felismerése

Az előző fejezetek bemutatták a névelemek és a tulajdonságok megkeresését a mondatban. Ezek is fontos lépések, de a célom az, hogy egy adott névelemhez tartozó tulajdonságokra vonatkozó érzelmeket határozzam meg. Első hallásra ez nem tűnik egy bonyolult feladatnak, de próbáljuk meg definiálni a problémát. Egy mondatban több névelem és tulajdonság is előfordulhat. A következő eseteket lehet megkülönböztetni:

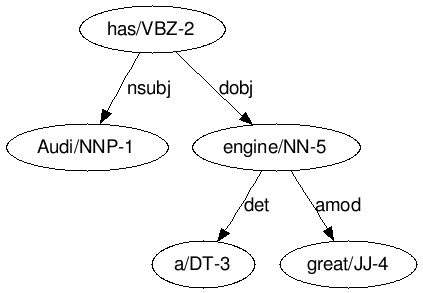
* Egy névelem, egy tulajdonság:
  + Példa: „This Audi has a great engine.”
  + A fenti mondatban a névelem az „Audi”, a tulajdonság pedig az „engine”.
* Egy névelem, több tulajdonság:
  + Példa: „The brake and engine of this Audi is great.”
  + A fenti mondatban a névelem az „Audi”, a tulajdonságok a „brake” és „engine”
* Több névelem, egy tulajdonság
  + Példa: „The seats in the VW and BMW are very comfortable.”
  + A fenti mondatban a névelemek a „VW” és „BMW”, a tulajdonság pedig a „seat”
* Több névelem, több tulajdonság
  + Példa: „BMW has a powerful engine, but VW has the best brakes.”
  + A fenti mondatban a névelemek a „BMW” és „VW”, a tulajdonságok pedig az „engine” és „brake”

Az előző eseteknél az is előfordulhat, hogy egy adott mondatban olyan névelem vagy tulajdonság fordul elő, mely az érzelemelemző rendszer számára ismeretlen. Például a „BMW has a powerful engine, but VW has the best brakes.” mondat esetében tételezzük fel, hogy a rendszer számára ismert a „BMW” névelem és a „brake” tulajdonság. Ebben az esetben nem létezik reláció az ismert névelem és tulajdonság között, mivel az ismert „brake” tulajdonság az ismeretlen „VW” névelemmel, valamint az ismeretlen „engine” az ismert „BMW” névelemmel áll relációban. Fontos, hogy az ilyen kivételes esetekben is a reláció felismerés a megfelelő eredményt adja.

Mielőtt ismertetném az algoritmust, be kell mutatnom néhány fogalmat a mondatban lévő szavak közti függőségek meghatározásáról. A Stanford Core NLP motor úgy volt tervezve, hogy egyszerű módon leírja a mondatokban fennálló nyelvtani kapcsolatokat, így azok is megérthetik, akiknek nem szakterületük a nyelvészet [7]. A mondatban lévő összes relációt egységesen tárolja, mint egy három egységből álló együttest: (reláció típus, első szó, második szó). Az „Audi has a great engine.” mondatban a következő relációkat határozza meg a Stanford Core NLP motor:

|  |  |
| --- | --- |
| **Függőségek** | **Stanford Core NLP Tankönyv leírás** [7] |
| nsubj(has-2, Audi-1) | Főnévi igenév, az ige és a főnév közti kapcsolat |
| root(ROOT-0, has-2) | Gyökér reláció, a mondat gyökere |
| det(engine-5, a-3) | Determináns, a főnevet meghatározó determináns és a főnév közti kapcsolat |
| amod(engine-5, great-4) | Melléknévi módosító, mely módosítja a főnév értelmét |
| dobj(has-2, engine-5) | Közvetlen tárgy, az ige közvetlen tárgya |

A fenti függőségeket gráfként tárolják. Az 3.1. ábra a fenti függőségeknek megfelelő gráfot mutatja:



3. ábra - Függőségek diagram

A Stanford Core NLP motor lehetővé teszi a függőségek összevonását. Az „Audi has a great engine under its hood.” mondatban a „prep(engine-5, under-6)” és „pobj(under-6, hood-8)” relációkat összevonva a következő relációt kapjuk: „prep\_under(engine-5, hood-8)”. Az összevonás alapját az képezi, hogy az „under”szó kapcsolatban áll egyszerre az „engine” és a „hood” szavakkal. A szövegelemző felismerve ezt modósítja a kapcsolat típúsát és a „prep” után fűzi a kapcsolatot teremtő kötőszót „prep\_under” és egybevonja a két összekötött szót „prep\_under(engine-5, hood-8)”.

A 3.2. ábra az alap- és összevont függőségeket mutatja.

|  |  |
| --- | --- |
| **(a)** | **(b)** |

4. ábra - Alap (a) és összevont (b) függőségek

A mondatban lévő függőségek bemutatása szükséges volt, mivel ebből indul ki a névelem – tulajdonság kapcsolatokat meghatározó algoritmus. Első lépésként a bemeneti mondatnak meghatározza a függőségi gráfját. Itt megjegyezném, hogy az algoritmus a függőségi gráfokat nem tekinti irányított gráfoknak. Az előző két fejezetből ismert, hogy a névelemek és a tulajdonságok minden esetben főnevek. Ezt a feltételezést felhasználva, meghatározom a távolságot a gráfban szereplő névelemek között. A gráfban az élek hosszát 1-nek tekintem. Ezt követően minden ismert névelemre az algoritmus meghatározza a hozzá legközelebb lévő ismert tulajdonsághoz vezető út távolságát. Ha a tulajdonsághoz nincs közelebb lévő névelem, mint az éppen aktuális névelem akkor sikeresen megtalálta a névelem – tulajdonság relációt. Ha a tulajdonsághoz más tulajdonság is vezet ugyanolyan távolságú út, mint az éppen aktuális névelemhez lévő távolság, akkor a névelem azokkal a tulajdonságokkal is relációban áll. A névelem – tulajdonság kapcsolatokat meghatározó algoritmust a következő pszeudókód mutatja be:

FindRelations(List<String> keywords, List<String> characteristics, String sentence)

**begin**

Map<String, List<Pair<String, Integer>>> distancesByNoun

List<Pair<String, String>> relations

graph = StanfordParser.CreateDependencyGraph(sentence)

List<String> nouns = FindAllNounsInGraph(graph)

**for** noun **in** nouns

distancesByNoun[noun] = FindDistancesToOtherNouns(noun, graph)

endfor

**for** (keyword, distances) **in** distancesByNoun

**if** (keywords.Contains(keyword))

distancesToCharacteristics = distances.Where(d => characteristics.Conatins(d.First))

Integer minDistanceToCharacteristics = distancesToCharacteristics.Min(d => d.Second)

**for** characteristic **in** distancesToCharacteristics.Where(d => d.Second == minDistanceToCharacteristics).Select(d => d.First)

List<Pair<String, Integer>> distancesFromCharacteristic = distancesByNoun[characteristic]

Integer minDistanceToKeywords = distancesFromCharacteristic.Where(d => keywords.Contains(d.First)).Min(d => d.Second)

List<String> closestNounsToCharacteristic = distancesFromCharacteristic.Where(d => d.Second == minDistanceToKeywords).Select(d => d.First)

**if** (closestNounsToCharacteristic.Contains(keyword))

relations.Add(Pair(keyword, characteristic))

**for** otherCharacteristic **in** closestNounsToCharacteristic.Where(n => n != keyword)

**if** (characteristics.Contains(otherCharacteristic))

relations.Add(Pair(keyword, otherCharacteristic))

**endif**

**endfor**

**endif**

**endfor**

**endif**

**endfor**

**return** relations

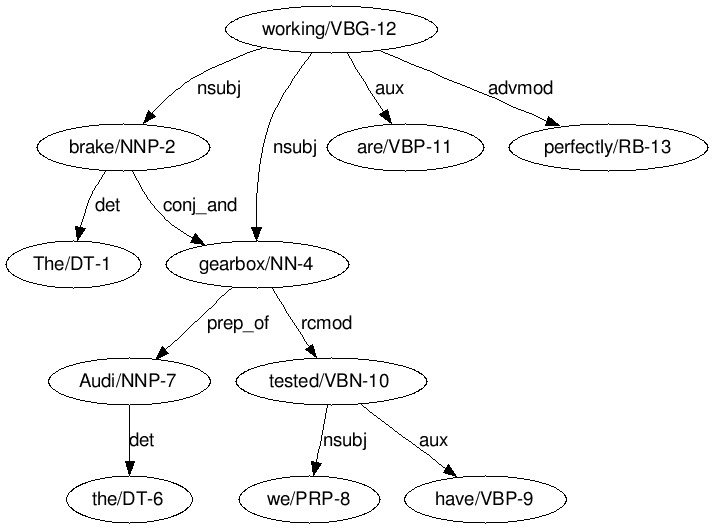
**end**

Az előbbi pszeudokódban a következő adatstruktúrákat és jelöléseket használtam:

|  |  |
| --- | --- |
| **Pszeudokód** | **Magyarázat** |
| Pair<String, Integer> p | Egy String-et és egész számot tartalmazó páros. p.First-el a String-et érjük el, p.Second-al az egész értéket |
| List<String> k | Egy String-eket tartalmazó lista |
| Map<String, Integer> m | Kulcs-érték párokat tartalmazó hashmap. A kulcs típusa String, az érték típusa egész szám |
| m[“kulcs1”] = 1 | Érték tárolása a hashmap-ben. |
| list.Where(d => d == 2) | Lista szűrése, azon elemeket adja vissza, egyenlők 2-vel. |
| list.Select(d => d.First) | Lista projekciója (mint adatbázistábláknál a SELECT). A visszaadott listában annyi elem lesz, mint az eredeti listában, az érték viszont minden elem esetében a d.First által tárolt érték lesz. |

A következőkben egy rövid példán szemléltetem az algoritmus működését. Legyen a következő mondat, melyben meg akarjuk határozni a névelem – tulajdonság kapcsolatokat: „The brake and gearbox of the Audi we have tested are working perfectly.”

A 3.3. ábra mutatja a mondat összevont függőségi gráfját.



5. ábra - Példamondat összevont függőségi gráfja

A relációkat meghatározó algoritmusnak megadjuk az „Audi” szót mint névelem, valamint a „brake” és „gearbox” szavakat mint tulajdonság. Első lépésként az algoritmus meghatározza a mondatban szereplő összes főnév közötti távolságokat. Ezt az 1. táblázat foglalja össze.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brake | Gearbox | Audi |
| Brake | - | 1 | 2 |
| Gearbox | 1 | - | 1 |
| Audi | 2 | 1 | - |

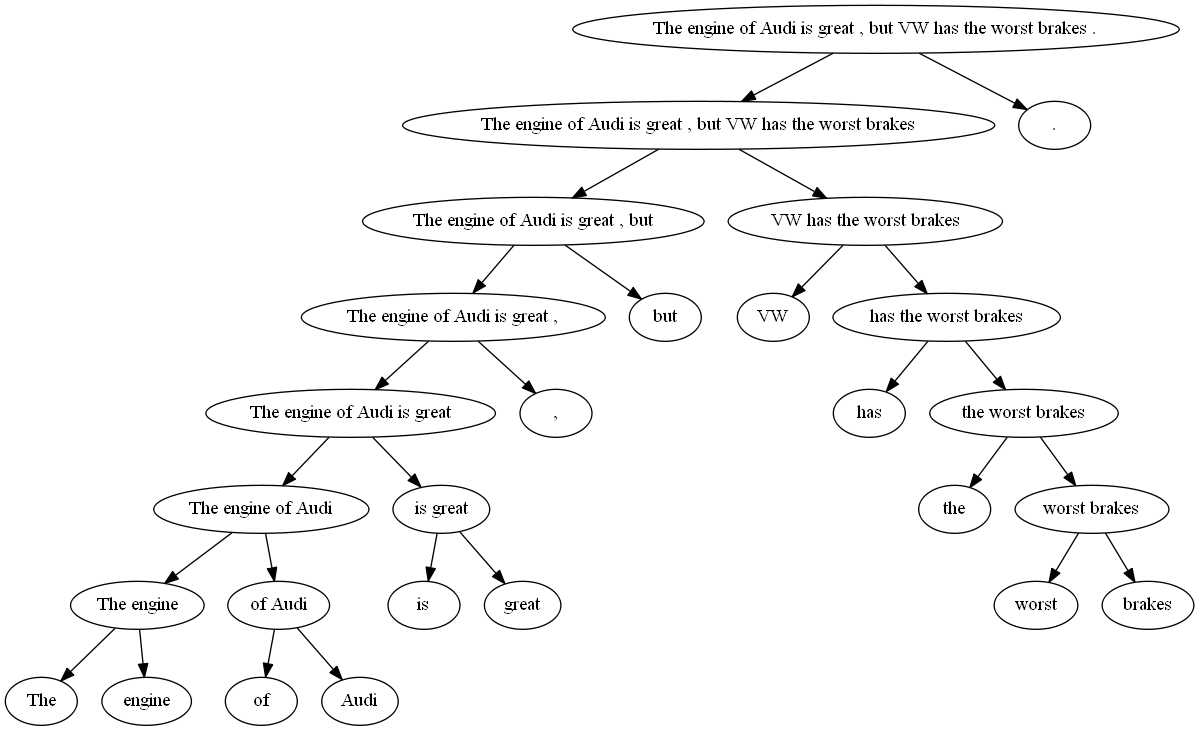
3. táblázat - Távolság mátrix

Második lépésként az algoritmus meghatározza az „Audi” névelemhez legközelebbi karakterisztika távolságát. Jelen esetben ez a „gearbox”. Következő lépésként az algoritmus megnézi, hogy létezik-e a „gearbox”-hoz közelebb lévő névelem, mint az „Audi”. Mivel ilyen névelem nem létezik, az algoritmus megtalálta az „Audi” és „gearbox” közt fennálló relációt. Utolsó lépésként az algoritmus megvizsgálja, hogy létezik-e a „gearbox”-tól más tulajdonsághoz vezető út, melynek hossza megegyezik 1-el (az „Audi” és „gearbox” közti távolsággal). Ebben az esetben létezik ez az út, így az algoritmus megtalálta az „Audi” és „brake” közt fennálló kapcsolatot is. Láthatjuk, hogy a fenti példa esetében az algoritmus a mondatban lévő mindkét kapcsolatot megtalálta. Az 5. fejezetben az eredmények bemutatásánál, részletesen kitérek majd az algoritmus hatékonyságára, melyet ún. „unit tesztek” segítségével, a fent említett minden esetre, több példamondaton keresztül teszteltem.

## Érzelem meghatározás

Az érzelem meghatározására a Stanford Core NLP motor érzelem-meghatározó komponensét használtam. A legtöbb érzelem-meghatározó rendszer úgy működik, hogy a szavakat külön-külön elemzik, pozitív pontokat adva a pozitív szavaknak, negatívat a negatívaknak, majd összesítik a pontokat [9]. Így figyelmen kívül hagyják a szórendet, ami bizony fontos információ. A Stanford érzelem-meghatározó rendszer egy rekurzív tenzor neuronhálót használ az érzelmek meghatározására és strukturálisan felépíti a mondatnak megfelelő érzelmi fát [10]. Ezt a hálót, egy érzelem fabankot (sentiment treebank) használva tanítottak. Az eddig létező érzelem-meghatározó rendszerekhez képest egy 5,4%-os növekedést mutatott a pontosságát tekintve, az egész mondatos érzelemelemzésnél. A részmondatos elemzéseknél 9,7%-al eredményesebb, mint a többi érzelem-meghatározó rendszer. Ugyanakkor ez az egyetlen olyan modell, mely képes felismerni a tagadást, valamint azt, hogy ez pontosan melyik részére vonatkozik a mondatnak. A Stanford érzelem-meghatározó rendszer a következő kategóriákba sorolja az érzelmeket: nagyon negatív, negatív, semleges, pozitív és nagyon pozitív. Az érzelmi fa minden csomópontjának megfeleltet egy ilyen ötös értéket, százalékosan kifejezve. Ez a fuzzy logikában szereplő lingvisztikus változókhoz való hovatartozáshoz hasonló megoldás, jelen esetben a lingvisztikus változók az érzelemkategóriák: pl. egy csomópont 10%-ban nagyon negatív, 12%-ban negatív, 8%-ban semleges, 42%-ban pozitív és 28%-ban nagyon pozitív.

Ahhoz, hogy meghatározzam a névelem – tulajdonság pároshoz kapcsolódó érzelmeket, felépítettem a Stanford Core NLP érzelem-meghatározója segítségével az egész mondatra vonatkozó érzelmi fát. A 3.4. ábra mutatja a „The engine of Audi is great, but VW has the worst brakes.” mondatnak megfelelő érzelmi fát. Ebben a mondatban két névelem: „Audi” és „VW”, valamint két tulajdonság van: „brake” és „engine”. A névelem – tulajdonság meghatározás során, a rendszerem megtalálja az „Audi” és „engine”, valamint a „VW” és „brake” relációkat. Ahhoz, hogy a relációnak megfelelő érzelmi értéket meg tudjuk határozni, a következő szabályt definiáltuk: azokat a csomópontokat választjuk az érzelmi fából, melyekben szerepel a névelem, a tulajdonság és egy ige. Az érzelem-meghatározó algoritmus minden reláció esetében poszt-order bejárást használva halad végig a csomópontokon. Az első olyan csomópontnak az érzelmi értéke felel meg az aktuális relációnak, mely teljesíti az előbbi szabályt.



6. ábra - Érzelmi fa

A 3.4. ábra esetében, ez a két csomópont a „The engine of Audi is great” és a „VW has the worst brakes”. A mutatja a Stanfor Core NLP érzelem-meghatározója által megadott érzelem értékeket.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Csomópont** | **Nagyon negatív** | **Negatív** | **Semleges** | **Pozitív** | **Nagyon pozitív** |
| „The engine of Audi is great” | 3% | 8% | 15% | 54% | 21% |
| „VW has the worst brakes” | 30% | 48% | 18% | 2% | 1% |

4. táblázat - Érzelem értékek

A következő pszeudokód írja le az érzelem-meghatározó algoritmust:

FindSentiment(List<Relation> relations, String sentence)

**begin**

SentimentTree = StanfordParser.ConstructSentimentTree(sentence)

List<SentimentNode> postOrder = SentimentTree.nodesInPostOrder()

List<Sentiment> sentiments

**for** r **in** relations

**for** node **in** postOrder

**if** node.text.Contains(r.Keyword) **AND** node.text.Conatain(r.Characteristic) **AND** node.HasVerb()

sentiments.Add(r, node.SentimentValues())

**endif**

**endfor**

**endfor**

**return** sentiments

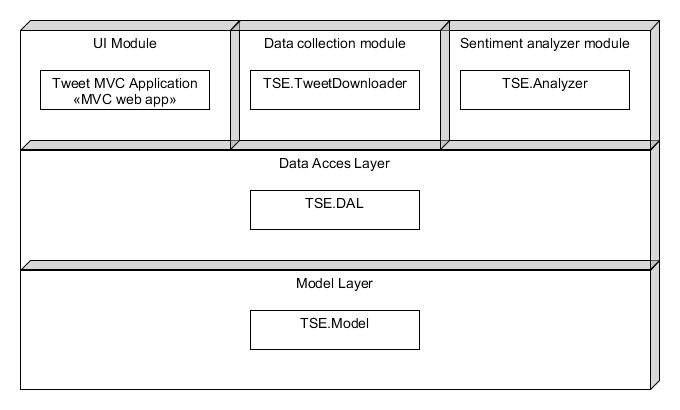
**end**

# A rendszer megvalósítása

## A rendszer specifikációi és architektúrája

Az általam elkészített rendszer tartalmaz egy web aplikációt, két szervíz komponenst, egy Data Acces Layer komponenst és egy model komponenst. A web aplikáción keresztül éri el a felhasználó az adminisztrációs és az eredményjelző modult. A két szervíz lehetőséget ad a tweetek letöltésének, valamint az érzelem elemzés elindítására és megállítására. A Data Acces Layer komponensben lévő osztályok segítségével olvassuk ki és módosítjuk az adatbázisban tárolt adatokat. A model komponens tartalmazza az adatbázisban szereplő tábláknak osztályokra való leképzését (ORM - object relational mapping).

* A rendszer moduljai a következők:
  + Tweet-gyűjtő modul
  + Érzelemelemző modul
  + Adminisztrációs modul
  + Eredményjelző modul



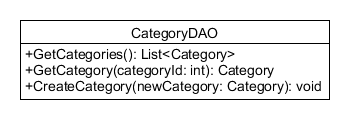
## A részletes tervezés

A következő alfejezetek külön-külön mutatják be a rendszer különböző moduljainak a megvalósítását.

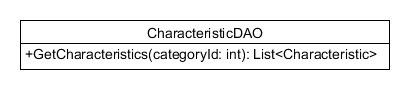
### DAL modul

A Data Acces Layer modul szerepe az aplikáció és adatbázis közti komunikáció megvalósítása. Ebben a modulban lévő osztályok segítségével az adatbázis tábláira végzünk lekérdezéseket, viszünk be új adatokat, modósítjuk a létező adatokat és törlünk az adatokból.

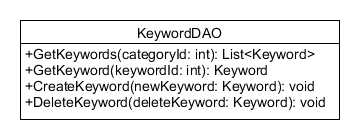
A CategoryDAO osztály az adatbázisban a Category táblához biztosít hozzáférést. A GetCategories metódus visszatéríti egy listában a tábla sorait. A GetCategory metódus visszatéríti a kapott paraméternek megfelelő Category elemet a táblából. A CreateCategory metódus beszúrja a táblába a paraméterként kapott Category típusú elemet.



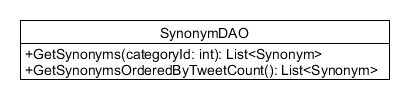
A CharacteristicDAO osztály az adatbázisban a Characteristic táblához biztosít hozzáférést. A GetCharacteristics metódus visszatéríti egy listában mely elemei Characteristic típusúak a paraméterként megkapott kategóriának megfelelő tulajdonságokat.



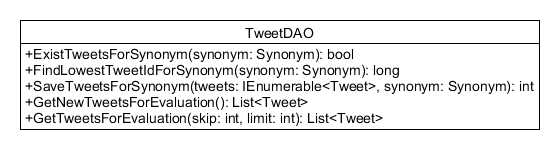
A KeywordDAO osztály az adatbázisban a Keyword táblához biztosít hozzáférést. A GetKeywords metódus visszatéríti egy Keyword típusú elemekből álló listában a paraméterként megkapott kategóriának megfelelő kulcsszavakat. A GetKeyword metódus visszatéríti azt a Keyword típusú elemet amely megfelel a paraméter értékének. A CreateKeyword metódus beszúrja a táblába a paraméterként kapott Keyword típusú elemet. A DeleteKeyword metódus letörli a táblából a paraméterként kapott Keyword típusú elemet.



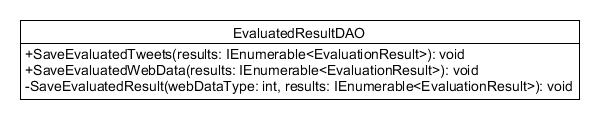
A SynonymDAO osztály az adatbázisban a Synonym táblához biztosít hozzáférést. A GetSynonyms metódus visszatéríti egy Synonym típusú elemekből álló listában a paraméterként megkapott kategóriának megfelelő kulcsszavakat. A GetSynonymsOrderedByTweetCount metódus visszatérít egy Synonym típusú elemekből álló listát mely rendezett az adott szinonimákat tartalmazó tweetek száma szerint.



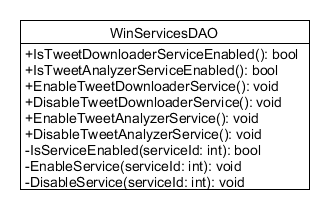
A TweetDAO osztály az adatbázisban a Tweet táblához biztosít hozzáférést. Az ExistTweetsForSynonym metódus igazat térít vissza ha léteznek a paraméterként kapott szinonimát tartalmazó tweetek a táblában, másképp hamis értéket térít vissza. A FindLowestTweetIdForSynonym a kapott paraméternek megfelelő tweetek közül annak a tweetnek az ID-ját téríti vissza amelyiknek a legkisebb az értéke. A SaveTweetsForSynonym beszúrja az adatbázisba a paraméterként megkapott lista elemeit és visszatéríti a sikeresen beszúrt tweetek számát. A GetNewTweetsForEvaluation visszatérít egy Tweet típusú elemeket tartalmazó listát amelyekben olyan tweetek szerepelnek, amelyekre még nem végeztünk érzelem elemzést. A GetTweetsForEvaluation metódus visszatérít egy Tweet típusú elemeket tartalmazó listát amelyben az ID szerint rendezett tweetekből megmondjuk, hogy hány tweetet szökünk át és legtöbb hány darab tweetet kérünk a listában. Erre szolgálnak a skip és limit paraméterek.



Az EvaluatedResultDAO osztály az adatbázisban az EvaluatedResult táblához biztosít hozzáférést. A SaveEvaluatedTweets és SaveEvaluatedWebData metódusok a SaveEvaluatedResult metódust felhasználva mentik le a táblákba az érzelmi elemzés eredményeit, melyek a tweetek, illetve weboldalakról származó adatok szövegeire voltak végrehajtva. A SaveEvaluatedResult metódus beszúrja a táblába a paraméterként kapott EvaluationResult típusú elemeket tartalmazó lista elemeit.



A WinServicesDAO osztály az adatbázisban az WinServices táblához biztosít hozzáférést. Az IsTweetDownoaderServiceEnabled metódus lekérdezi az IsServiceEnabled metodús segítségével, hogy az adatbázisban engedélyezve van vagy nincs a tweet letöltési szervíz és ennek megfelelően igaz vagy hamis értéket térít vissza. Az IsTweetAnalyzerServiceEnabled metódus ugyanazt végzi az érzelemelemző szervízre. Az EnableTweetDownloaderService és EnableTweetAnalyzerService metódusok az EnableService metódus segítségével az adatbázisban igazra állítják a megfelelő szervíz engedélyezésének az értékét. A DisableTweetDownloaderService és DisableTweetAnalyzerService metódusok a DisableService metódus segítségével az adatbázisban hamisra állítják a megfelelő szervíz engedélyezésének az értékét. Az IsServiceEnabled metodús visszatéríti, hogy a paraméterként kapott szervíz engedélyezve van vagy nincs az adatbázisban. Az EnableService metódus a paraméterként kapott szervíz engedélyezési értékét igazra állítja az adatbázisban. A DisableService metódus a paraméterként kapott szervíz engedélyezési értékét hamisra állítja az adatbázisban.



### Tweet-gyűjtő modul

A microblogging napjainkban egy nagyon népszerű komunikációs eszközzé vált az internetezők között. [11] Felhasználok milliói osztják meg véleményüket a mindennapi élet különböző területeiről. Épp ezért a microblogos web-oldalak gazdag adatforrást jelentenek vélemények bányászatához és érzelmi elemzéséhez. A fontosabb okok amiért Tweetert érdemes elemezni a következők:

* különböző személyek különböző topikokról nyilvánitanak ki véleményeket, tehát egy értékes vélemény-forrást képez; az adathalmaz amit tartalmaz óriási és naponta növekedik;
* a twitter felhasználó típusai változatosak, átlagos felhasználoktól celebekig, cégek, politikusok, még államelnökök is, tehát változatos csoportok véleményével találkozunk különböző szociális és érdekképviseleti csoportoktól;
* a felhasználok különböző országokból vannak.
* Hogyan működik?

A twitter.com egy online szociális hálozat melyet milliók használnak a föld különböző országaiból kapcsolat-tartásra családtagokkal, barátokkal, munkatársakkal számítógépen vagy mobil-eszközökön keresztül. A felhasználói felület rövid szöveges üzenetek posztolását teszi lehetővé (140 karakter maximum hosszal) amelyek láthatóak bármely más Twitter felhasználó számára. A felhasználok megjelölhetik azokat a személyeket akiket szeretnének követni, ez esetben a felhasználó értesítve lesz mikor a követett új üzenetet posztolt. A követett-nek jelölt felhasználó nem feltétlenül kell vissza-kövesse azt a személyt aki őt követi, ily modón a twitter egy irányitott kapcsolatú szociális hálót alkot.

* Tweet felépítése

A tweet az elemi parányi építő-tömbje a Twitternek. Felépítését a következő táblázatban mutatom be:

|  |  |  |
| --- | --- | --- |
| Mezőnév | Típus | Leírás |
| annotations | Object | Nem használt. A jövőben az állapot annotálására fog szolgálni. |
| contributors | Collection  of Contributors | Nullázható. Azokat a felhasználókat tartalmazza akik hozzájárultak a tweet-hez a fő-tweetszerkesztő mellett. |
| coordinates | Coordinates | Nullázható. A tweet földrajzi helyét mutatja. |
| created\_at | String | UTC idő, a tweet létrehozásának az ideje. |
| current\_user\_retweet | Object | Tartalmazza a tweet ID-t a felhasználó saját re-tweetjének az adott tweetről. |
| entities | Entities | Entitások amik a tweet szövegjéből lettek kiszűrve, mint pl.hashtagek, url-k. |
| favorite\_count | Integer | Nullázható. Hány felhasználó jelölte meg kedvencként a tweetet. |
| favorited | Boolean | Nullázható. A hitelesitő felhasználó megjelölte mint kedvenc vagy sem. |
| filter\_level | String | A maximum szűrési szint amelyen a tweet áramlani fog. |
| geo | Object | Érvénytelenített. Helyette használjuk a coordinates mezőt. |
| id | Int64 | A tweet egyedi azonósítoja, egész számként. |
| id\_str | String | A tweet egyedi azonósítoja, karakterláncként. |
| in\_reply\_to\_screen\_name | String | Nullázható. Ha egy retweetről van szó, akkor az eredeti tweet szerzőjének a felhasználó nevét tartalmazza. |
| in\_reply\_to\_status\_id | Int64 | Nullázható. Ha egy retweetről van szó, akkor az eredeti tweet ID tartalmazza egész számként. |
| in\_reply\_to\_status\_id\_str | String | Nullázható. Ha egy retweetről van szó, akkor az eredeti tweet ID tartalmazza karakterláncként. |
| in\_reply\_to\_user\_id | Int64 | Nullázható. Ha egy retweetről van szó, akkor az eredeti tweet szerzőjének az ID számát tartalmazza egész számként. |
| in\_reply\_to\_user\_id\_str | String | Nullázható. Ha egy retweetről van szó, akkor az eredeti tweet szerzőjének az ID számát tartalmazza karakterláncként. |
| lang | String | Nullázható. A számítógép által felismert nyelv BCP 47 kódját tartalmazza. |
| place | Places | Nullázható. A tweethez hozzáfűzödő helység neve és földrajzi koordinátái. |
| possibly\_sensitive | Boolean | Nullázható. Jelzi ha a tweet szövege tartalmaz url-t. |
| quoted\_status\_id | Int64 | Ha a tweet egy idézet-tweet, ez a mező tartalmazza az idézett tweet ID számát egész számként. |
| quoted\_status\_id\_str | String | Ha a tweet egy idézet-tweet, ez a mező tartalmazza az idézett tweet ID számát karakterláncként. |
| quoted\_status | Tweet | Ha a tweet egy idézet-tweet, ez a mező tartalmazza az idézett tweetet mint Tweet objektum. |
| scopes | Object | Kulcs-érték párosok, amelyek a kurrens tweet leközlési szándékának a kontextusát jelzi. |
| retweet\_count | Int | Hányszor volt re-tweetelve az adott tweet. |
| retweeted | Boolean | Volt-e retweetelve az adott tweet hitelesitett felhasználó által. |
| retweeted\_status | Tweet | Az eredeti tweet objektumát tartalmazza egy re-tweetelt tweet esetében. A többszöri re-tweetek esetében a köztes re-tweet is az eredetit fogja mutatni. |
| source | String | A tweet posztolása HTML formátumként. |
| text | String | A tweet UTF 8 szövegje. |
| truncated | Boolean | Mutatja ha a tweet text mezője csonkított-e, azaz meghaladta a 140 karakterszámot. |
| user | Users | A felhasználó aki posztolta az adott tweetet. |
| withheld\_copyright | Boolean | Az adott tweet tartalma szerzői joggal védett a Digital Millennium Copyright Act értelmében. |
| withheld\_in\_countries | Array of String | Melyik országokban az adott tweet tartalma szerzői joggal védett a Digital Millennium Copyright Act értelmében. |
| withheld\_scope | String | A szerzői jog védelme mire vonatkozik, az állapotra vagy a felhasználóra. |

* Twitter API

A disszertácós dolgozatom elkészítéséhez a Twitter API-k közül a REST API-t használtam, az aplikáció tovább fejlesztéseként érdemes lenne használni a Streaming API-t is.

A REST API bíztositja a programokon keresztüli hozzáférést, olvasást és írást, a Twitter adataihoz. Létrehozhatunk új Tweet-et, olvashatjuk a felhasználok profilját és a követők adatait, egyedi kereséseket hajthatunk végre és sok minden egyebet. A REST API a felhasználók és aplikácíok azonosítása az OAuth keresztül történik, a válaszok formátuma pedig JSON.

A Search API része a Twitter 1.1v REST API-nak. Ez az API lekérdezéseket biztosít a friss vagy népszerű Tweetekből és viselkedése hasonlít, de nem azonos a Twitter mobil vagy web kliensek Search funkciójával, mint például Twitter.com search. Fontos tudni azt, hogy a Search API fő célja a releváncia és nem a teljesség, azaz néhány Tweet és felhasználó lehet hiányozni fog keresések futtatásakor. Ha a fő célunk a teljesség akkor használhatjuk a Search API helyett a Streaming API-t. A Search API limitálja a lekérdezések mértékét. Jelenleg, mint felhasználó 180 kérés/lekérdezés 15 percnyi időtartamra. A csak-aplikáció azonositást használva, egy adott aplikáció 450 lekérdezés/kérés 15 percnyi időtartamra a határ, aplikáción belül felhasználóra való tekintet nélkül.

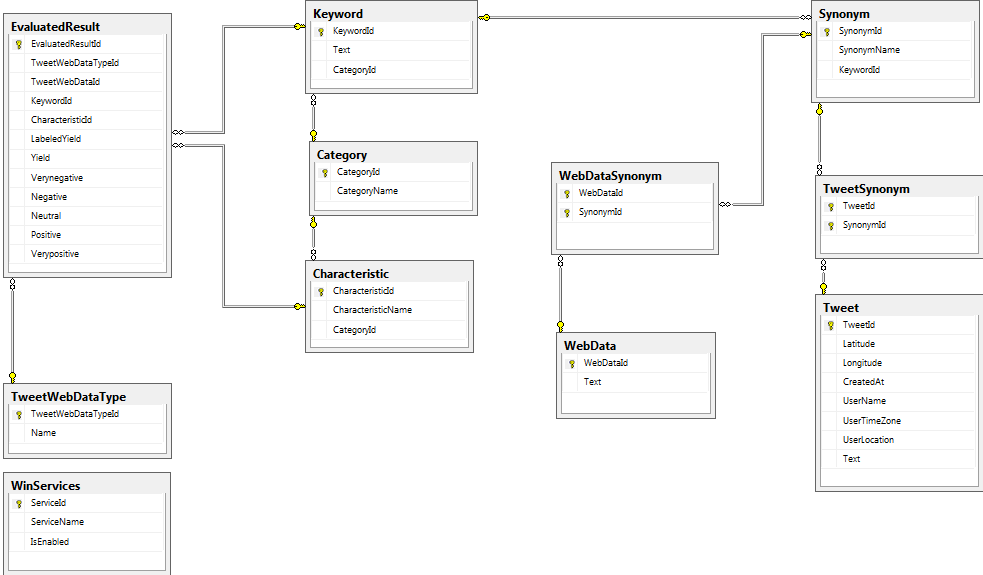
A Streaming API alacsony lappangású hozzáférést ad a Twitter globálisan streamelt adataihoz. Egy megfelelően implementált stream kliens üzeneteket küld ahogy Tweetek jelennek meg vagy egyéb események történnek. A Twitter három típusú folyam célpontot kínál:

* Publikus folyamok – A publikus adatok Twitteren keresztüli áramlatának folyama. Megfelel ha bizonyos felhasználókat vagy topikokat szeretnénk követni, vagy adatbányászatnak.
* Felhasználó folyamok – Egy-felhasználó folyam, tartalmazza az összes adatokat ami a Twitteren megfelel egy felhasználó szempontjának.
* Oldal folyamok – A több-felhasználó változata a felhasználó folyamoknak. Az oldal folyamok azokat a szervereket célozzák amelyek a több felhasználó nevében kell kapcsolodjanak Twitterhez.

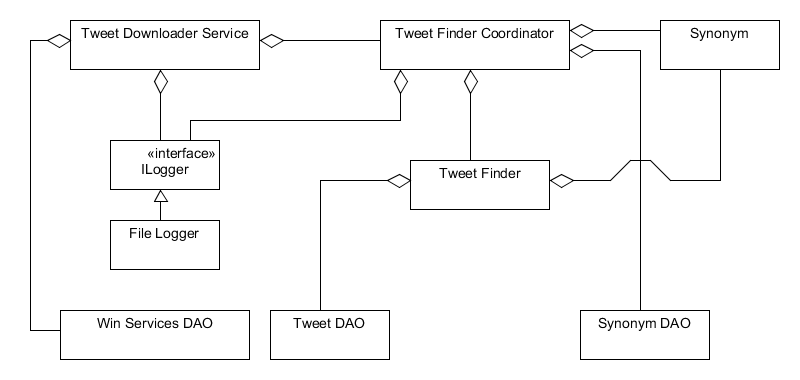
A következő táblázat bemutatja az adatbázisban szereplő táblákat, melyekbe az adatokat mentjük.

|  |  |
| --- | --- |
| Táblanév | Magyarázat |
| Category | Tartalmazza a kategóriákat. Pl. Járművek, politikusok, teniszezők,... |
| Keyword | Tartalmazza a kulcsszavakat a megfelelő kategoriáknak.  Pl. Audi, VW, Ford, Mercedes (járművek) |
| Characteristic | Tartalmazza a tulajdonságokat a megfelelő kategoriáknak.  Pl. Motor, kapcsolószekrény, kényelem (járművek) |
| Synonym | Tartalmazza a szinonimákat a megfelelő kulcsszavaknak.  Pl. A4, A6, Quatro (Audi) |
| TweetWebDataType | Tartalmazza a letöltött adatok forrás-tipusát. |
| Tweet | Tartalmazza a letöltött Tweeteket. |
| TweetSynonym | Segéd-tábla amely párositja a letöltött tweeteket és a tweet szövegében szereplő szinonimákat. |
| WebData | Tartalmazza a web-oldalakról letöltött adatokat. |
| WebDataSynonym | Segéd-tábla amely párositja a web-oldalakról letöltött adatokat és a ezeknek a szövegében szereplő szinonimákat. |
| EvaluatedResult | Tartalmazza az érzelmi kiértékelések eredményeit. |
| WinServices | Tartalmazza az aplikáció által futtatott szervizek állapotát.  Pl. Tweet Downloader, TweetStreamDownloader, Sentiment Analyzer |

A következő ábrán látható az adatbázis egyed-kapcsolat diagrammja:



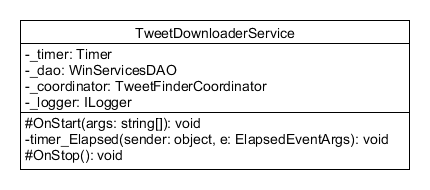
A Tweet-gyűjtést egy külön projekt látja el, melynek szerepe a Twitter-ről letölteni azokat a tweeteket amelyek megfelelnek a keresési kritériumoknak és ezeknek az elmentése saját adatbázisba. Ez a projekt szervízként fut az applikáción belül és az adminisztrációs felületről indítható el illetve állítható meg.



A Tweet gyűjtési projekt osztályai a követkekzők:

**TweetDownloaderService**  osztály a ServiceBase osztályt egészíti ki, szerepe a tweetek letöltési folyamatának az elindítása és megállítása. A \_timer adattaggal időzítjük azt az időközt amikor leellenőrizzük az adatbázisban a letöltés engedélyezve van-e vagy nem. A \_dao adattag biztosítja a kapcsolatot az adatbázisban lévő táblával amelyben tároljuk a tweet gyűjtési folyamat engedélyezését. A \_coordinator adattag segítségével fogjuk a tweet letöltési folyamatát elindítani vagy megállítani. A \_logger adattag szerepe a tweet gyűjtési folyamat során egy fájlba feljegyezni a végrehajtott műveleteket és ezeknek az eredményeit.

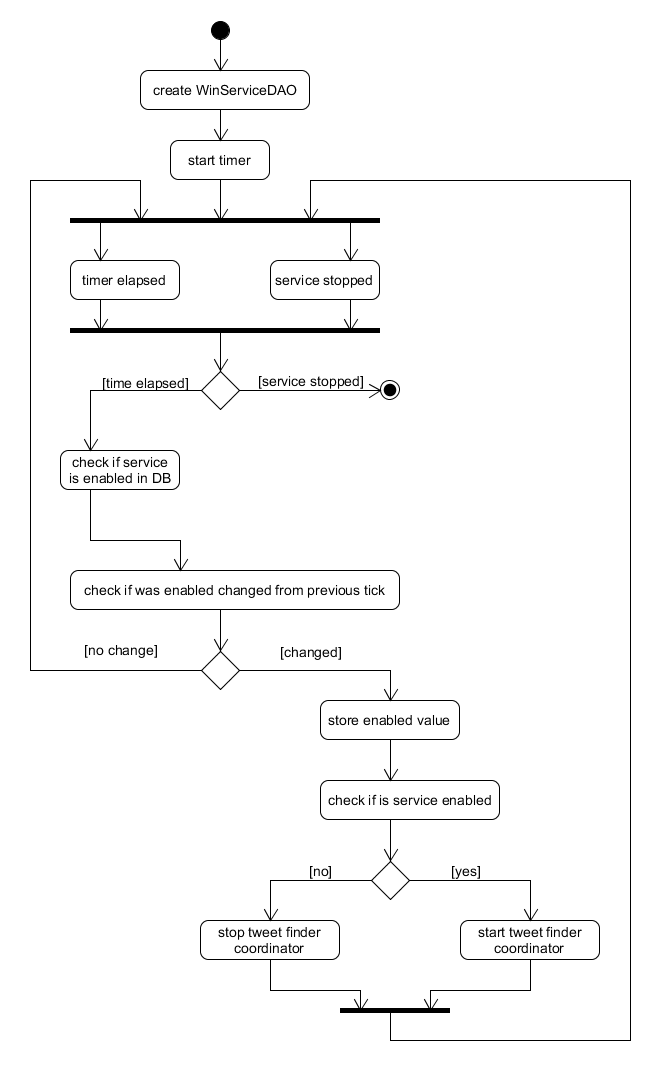
Az OnStart metodús inicializálja a \_timer,\_dao,\_logger és \_coordinator adattagokat és a \_timer időzítő segítségével meghívja a beállított időközönként a timer\_Elapsed metodúst. A timer\_Elapsed metodús a \_dao adattagon keresztül kiolvassa az adatbázisból, hogy a tweet gyűjtés engedélyezve van-e vagy nem, ezt elmenti egy változóba, majd az újabb időközönként ellenőrzi ha változott az engedélyezés állapota, ha változás történt akkor az engedélyezés állapotának függvényében loggolja a kapott beállításnak az állapotát és a \_coordinator adattagon keresztül elindítja vagy megállítja a tweet gyűjtési folyamatot. Az OnStop metodús a \_coordinator adattagon keresztül leállítja a tweet gyűjtési folyamatot.



A szervíz mükődési folyamata a következő: létrehozza a kapcsolatot az adatbázissal, utána elindít egy időzítőt amely 10 másodpercenként ellenőrzi az adatbázisban, hogy a letöltési szervíz engedélyezve van-e vagy nem és ha ennek az állapota megváltozott az elöző állapotához képest.

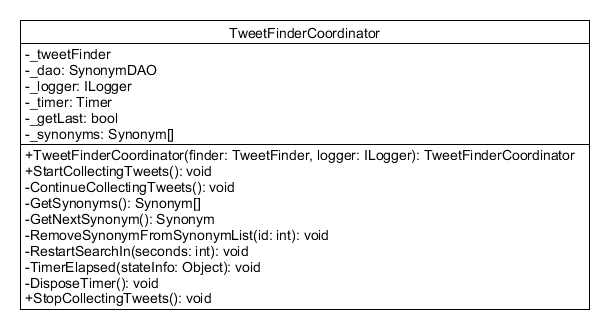
Ha nem volt változás akkor 10 másodperc múlva újraindul az adatbázisos ellenőrzés, ha változás volt akkor a letöltési folyamat engedélyezési állapot szerint elindítja a letöltést vagy megállítja.

A x.x ábra mutatja a TweetDownloaderService osztálydiagramját.

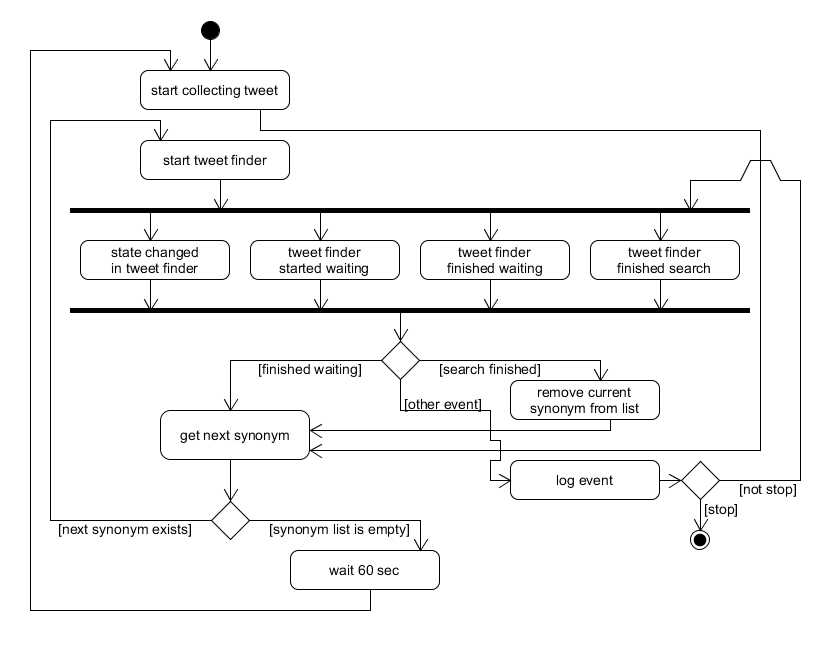
****

**TweetFinderCoordinator** szerepe a tweetek keresési és lementési folyamatának a vezérlése. A \_tweetFinder adattagon keresztül fogjuk elérni a TweetFinder osztály metodúsait. A \_dao adattag segítségével hajtunk végre majd lekérdezést az adatbázis synonym táblájáról. A \_logger adattag segít a folyamatok naplózásában. A \_timer adattaggal időzítjük a tweet keresés újraindítását. A \_getLast adattag mondja meg, hogy a szinonimák listájából az utolsó elemet vegyük ki vagy az elsőt. A \_synonyms adattag tartalmazza a keresés alapjául szolgáló szinonimák listáját.

A TweetFinderCoordinator osztály konstruktorában inicialízáljuk a \_dao adattagot valamint injektáljuk a \_logger és a \_tweetFinder adattagokat. A StartCollectingTweets meghívja a StopCollectingTweets metodúst, feltölti a \_synonyms adattagot a GetSynonyms metodús meghívásával majd meghívja a ContinueCollectingTweets metodúst. A ContinueCollectingTweets metodús a GetNextSynonym metodússal lekérdezi egy lokális változoba a következő keresett szinonimát. Ha a változó értéke null, akkor naplozza az információt, hogy nem talált keresésre szánt szinonimát, és meghívja a RestartSearchIn metodúst a 60 értékű paraméterrel. Ha nem null a keresett szinonima értéke, akkor naplozza az információt, hogy a keresést elinditotta az adott szinonomára és meghívja a \_tweetFinder adattagról a StartSearchAsync metodúst átadva a szinonima értékét paraméterként. A GetSynonyms metodús a \_dao adattagnak megprobálja meghívni a GetSynonymsOrderedByTweetCount metodúsát, ha ez sikerült akkor visszatéríti a szinonimák listáját, ha nem sikerül, akkor naplozza az információt, hogy a szinonima lista betöltése nem sikerült és null értéket térít vissza. A GetNextSynonym metodús abban az esetben ha a \_synonyms lista üres vagy nincs eleme null értéket térít vissza. Ha a lista nem üres akkor a \_getLast adattag értéke függvényében, ha igaz akkor az utolsó elemet, ha hamis akkor az első elemet téríti vissza. A RemoveSynonymFromSynonymsList paramáterként kapott szinonim azonosító értéket megkeresi a \_synonyms listában. Ha megtalálta és a \_synonyms lista nem üres akkor törli a listából. A RestartSearchIn metodús inicialízálja a megkapott parameter értékével a \_timer adattagot. A TimerElapsed metodús meghívja a DisposeTimer metodúst, le nullázza a \_synonyms adattag értékét majd meghívja a StartCollectingTweet metodúst. A DisposeTimer metodús felszabadítja a \_timer adattag által lefoglalt erőforrásokat és az értékét nullra állitjuk. A StopCollectingTweets metodús szintén meghívja a DisposeTimer metodúst és a \_tweetFinder adattagról a StopSearch metodúst.



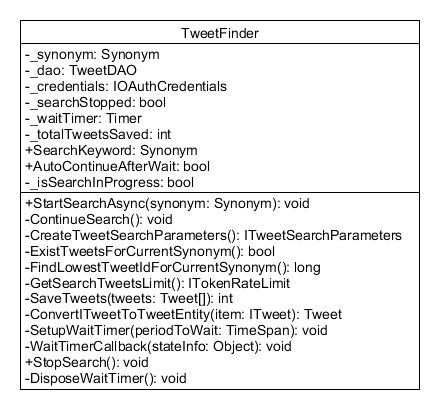
A TweetFinderCoordinator folyamat első lépésként megkeresi a soron következő szinonimát, ha létezik, akkor elindul a keresés, ha üres a szinonimák listája akkor várakozik 60 másodpercet és újraindítja a folyamatot. A keresési folyamat a következő lépéseiben több esetet kell lekezeljen mint változás történik a keresés állapotában, a kereső elkezdett várakozni, a kereső befejezte a várakozást vagy a kereső befejzte a keresést. Ha a kereső befejezte a várakozást akkor a folyamat a következő szinonim lekérdezésével folytatodik. A keresés befejezése után eltávolítjuk a szinonima listájából a keresett szinonimát és folytatjuk a következő szinonima lekérdezésével. Ha egyéb esemény következik be, akkor naplozzuk az eseményt és megállitsuk vagy újrakezdjük a folyamatot.

****

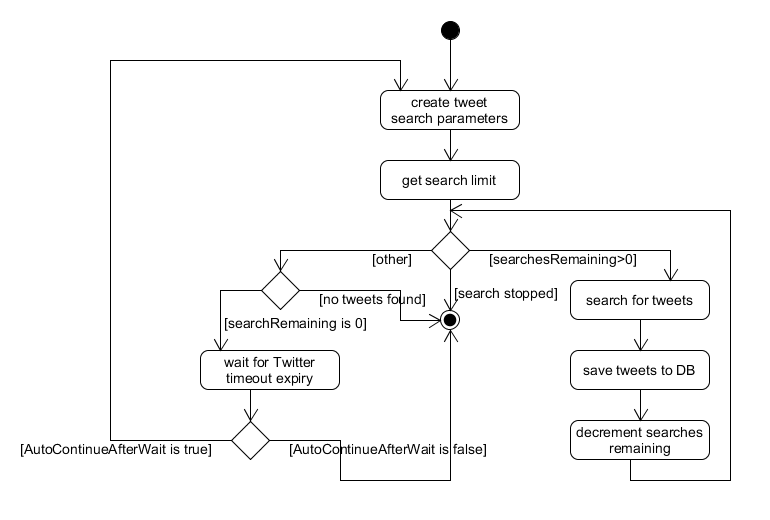
**TweetFinder** osztály szerepe a tweetek keresése, letöltése és elmentése az adatbázisba. A \_synonym adattag tárolja a keresett szinonimát. A \_dao adattag biztosítja a kapcsolatot az adatbázissal. A \_credentials adattag tartalmazza a Twitter API által követelt azonositót. A \_searchStopped adattag mutatja a keresés befejezését. A \_waitTimer adattag követi a Tweet API által megszabott letöltési korlátok időtartamát és szükség esetén várakoztatja a letöltési folyamatot. A \_totalTweetsSaved adattag tárolja a letöltött tweetek számát. A SearchKeyword adattag tárolja a keresett szinonimát. Az \_AutoContinueAfterWait adattag határozza meg, hogy automatikusan folytassa a tweetek keresését vagy sem. Az \_isSearchInProgress adattaggal kövessük, hogy van-e keresés folyamatban.

A StartSearchAsync metodús megkapja paraméterként a keresett szinonimát, meghívja a StopSearch metodúst, átadaja a paraméter értékét a \_synoynm adattagnak, igaz értéket ad az IsSearchInProgress adattagnak, hamis értéket ad \_searchStopped adattagnak, zéro értéket ad a \_totalTweetsSaved adattagnak, meghívja a DisposeWaitTimer metodúst, jelzi az OnSearchStarted esemény beköveztét és meghívja a ContinueSearch metodúst. A ContinueSearchAsync metodús előállítja a keresési paramétereket a CreateTweetSearchParameters metodús meghívásával, lekérdezi a Tweet API korlátozásait a GetSearchTweetsLimit metodússal, beállítva a searchesRemaining változó értékét a megengedett letölthető tweetek számara, majd jelzi az OnStateChanged eseménnyel a searchesRemaining értékét. Hiba fellépése esetén jelzi az OnErrorOccuredDuringSearch eseménnyel a hiba jellegét és várakoztatja a folyamatot egy percet. Ha a searchesRemaining nagyobb mint zéro, akkor a TweetInvi csomagot használva beállítsuka twitter API-hoz szükséges azonosítonkat és meghívjuk a tweet API által biztositott keresési metodúst. A letöltött tweetek egy ITweet típusú elemekből álló listába kerülnek. Csökkentjük a searchesRemaining változonk értékét és naplozzuk a letöltött tweetek számát valamint a searchesRemaining értékét. Ha a hiba lép fel akkor lenullázzuk a tweets listát, zéro értéket adunk a searchesRemaining változónak és naplozzuk a hiba jellegét. Ha a tweets listában léteznek elemek, akkor meghívjuk a SaveTweets metodúst, frissitjük a \_totalTweetsSaved adattagot és naplozzuk az adatbázisba sikeresen lementett tweetek számát. Ez a keresés és letöltési folyamat mindaddig ismétlödik míg a searchesRemaining változó értéke nagyobb mint zéro és a \_searchStopped adattag értéke hamis és a letöltött tweet listában vannak elemek. Ha a searchesRemaining változó értéke zéro lesz, akkor naplozzuk, hogy az idő lejárt és meghívjuk a SetupWaitTimer metodúst átadva paraméterként a várakoztatási idő értékét. Ha a \_searchStopped értéke igaz lesz, akkor \_isSearchInProgress adattagnak az értékét hamisra állítjuk és naplozzuk, hogy a keresés leállt. Ha az előbbi két eset egyike sem következik be akkor azt jelenti, hogy nem találtunk több tweetet amely teljesítené a keresési feltételeinket. Ebben az esetben az \_ isSearchInProgress adattag értékét hamisra állítjuk, naplozzuk az eseményeket, hogy több tweetet nem találtunk és összesen hány tweetet mentettünk le.

A CreateTweetSearchParameters eljárás készíti el a Tweet Search API – nak szükséges paraméter beállitását, megadva a keresett szinonimát, a keresett Tweetek nyelvét, milyen típusú Tweetet keresünk (friss, népszerű vagy vegyes) , a visszatérített eredmények maximális számát. Odafigyel és lekezeli egy adott Tweet folytatólagos letöltését követve az adott szinonimára talált Tweetek ID-ját. Erre a célra szolgál az ExistTweetsForCurrentSynonym eljárás amely megmondja, hogy léteznek-e letöltött Tweetek az adatbázisban a keresett szinonimáról. Abban az esetben ha már vannak letöltött Tweetek az adatbázisban a keresett szinonimáról akkor megkeressük a legkissebb ID – ju Tweetet és az ennél kisebb ID Tweetekkel folytatjuk a keresést. Ezt a feladatot szolgálja a FindLowestTweetIdForCurrentSynonym eljárás. A GetSearchTweetsLimit eljárás visszatéríti a Tweet Search API által megszabott letöltési korlátokat, hány keresést végezhetünk és mennyi az időkorlát amíg egy keresés tarthat. A SaveTweets eljárás perszisztálja felhasználva a TweetDAO projekt osztályait a letöltött Tweeteket. Mivel a letöltött Tweetek típusa Itweet szükséges átalakítani az aplikációnk által használt Tweet entitás típussá, erre használom a ConvertITweetToTweetEntity eljárást. A StopSearch eljárás megállítja a keresési folyamatot és meghív egy másik eljárást, DisposeWaitTimer-t, amely felszabadítja az időzítőt.

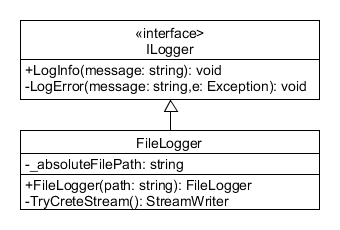


A TweetFinder osztály folyamatja a tweet keresési paraméter létrehozásával kezdődik. Ezután lekérdezzük a Tweet API-tól a számunkra érvényes keresési korlátokat. Egy három elágazású csomópont következik. Az első elagazás, ha a kersés leállt, ebben az esetben a folyamat leáll. A második elagazás, ha még kereshetünk tweeteket, azaz nem értük el a korlatot, akkor keresünk tweeteket, lementsük az adatbázisba, csökkentsük a maradt keresési korlátot és újrakezdjük a folyamatot. A harmadik elagazás, ha egyéb, azaz a mésik két eset közül egyik sem, ekkor ha már nem kapunk több tweetet a keresést megállítjuk. Ha még nem értük el a kersési korlátot akkor megvárjuk a korlátolt idő lejártát és annak függvényében, hogy az automatikus folytatási adattagnak igaz vagy hamis értéke van, újrakezdjük a keresést vagy megállítjuk a keresést.



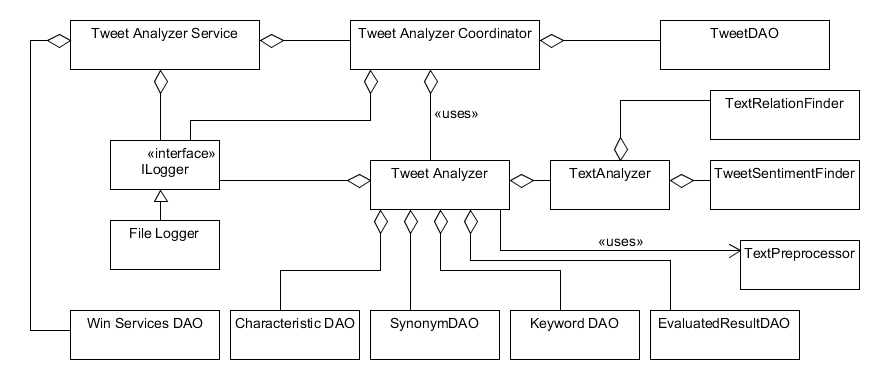
A FileLogger osztály szerepe a tweet gyűjtési folyamat során fellépő események naplozása.

A FileLogger osztály implementálja a ILogger interfészt. Az \_absoluteFilePath addattag tárolja annak a fájlnak a nevét és útvonalát amelybe a naplózás történik. A FileLogger konstruktora, paraméterként megkapja az útvonalat és ezt az értéket átadja az \_ absoluteFilePath adattagnak. A TryCreateStream eljárás megprobál kiírni a megkapott útvonalú fájlba és ha sikerül, akkor visszatérít egy stream-et. Az interfész két eljárásának az implementálásában a visszatérített stream-et felhasználva naplozzuk az eseményeket vagy a fellépő hibákat.



### Érzelemelemző modul

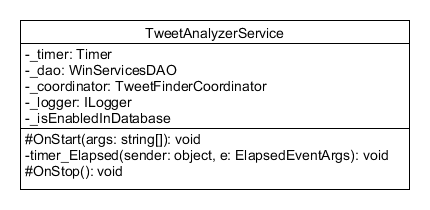
A Tweet-érzelemelemző modult egy külön projekt látja el, melynek szerepe a letöltött tweetek szövegének a feldolgozása és érzelem-elemzése. Ez a projekt szervízként fut az applikáción belül és az adminisztrációs felületről indítható el illetve állítható meg.



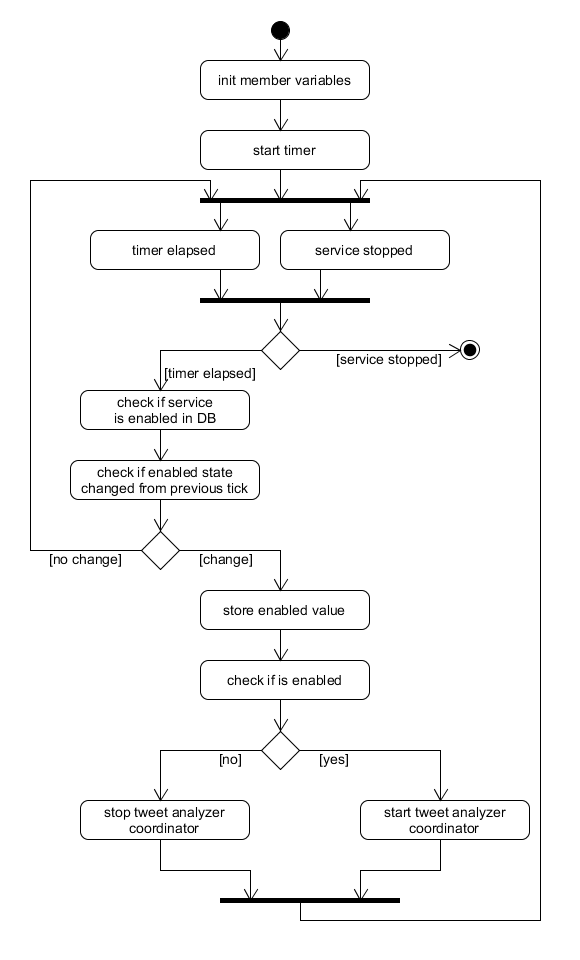
A Tweet elemző projekt a következő osztályokat tartalmazza:

**TweetAnalyzerService** osztály a ServiceBase osztályt egészíti ki, szerepe a tweetek letöltési folyamatának az elindítása és megállítása. A \_timer adattaggal időzítjük azt az időközt amikor leellenőrizzük az adatbázisban a letöltés engedélyezve van-e vagy nem. A \_dao adattag biztosítja a kapcsolatot az adatbázisban lévő táblával amelyben tároljuk a tweet érzelem elemző folyamat engedélyezését. A \_coordinator adattag segítségével fogjuk a tweet érzelem elemző folyamatát elindítani vagy megállítani. A \_logger adattag szerepe a tweet érzelem elemzési folyamat során egy fájlba feljegyezni a végrehajtott műveleteket és ezeknek az eredményeit. Az \_isEnabledInDatabase adattag értéke tükrözi az adatbázisban szereplő érzelem elemző folyamat státuszát, futhat az érzelem elemző szervíz vagy nem.

Az OnStart metodús inicializálja a \_timer,\_dao,\_logger és \_coordinator adattagokat és a \_timer időzítő segítségével meghívja a beállított időközönként a timer\_Elapsed metodúst. A timer\_Elapsed metodús a \_dao adattagon keresztül kiolvassa az adatbázisból, hogy a tweet érzelem elemzés engedélyezve van-e vagy nem, ezt elmenti egy változóba, majd az újabb időközönként ellenőrzi ha változott az engedélyezés állapota, ha változás történt akkor az engedélyezés állapotának függvényében loggolja a kapott beállításnak az állapotát és a \_coordinator adattagon keresztül elindítja vagy megállítja a tweet érzelem elemzési folyamatot. Az OnStop metodús a \_coordinator adattagon keresztül leállítja a tweet érzelem elemzési folyamatot.

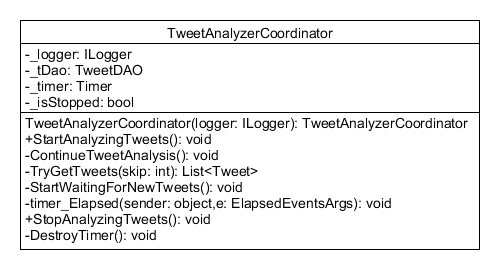


A szervíz mükődési folyamata a következő: inicializálja az adattagokat és elindítja az időzitőt. Ezután két eset lehetséges: lejárt az időzített periodús vagy megállították a szervízt. Ha megállították a szervízt akkor a érzelem elemző folyamat megáll. Ha az időzített periodús jért le akkor leellenőrizzük az adtbázisban a szerviz státuszát és megnézzük, hogy ez a státusz változott-e az előző időzítéshez képest. Ha nem változott akkor újrakezdjük a folyamatot. Ha változott a státusz akkor lemetjük az új státusz értéket. Ha a státusz értéke engedélyezett akkor elindítjuk a tweet analyzer coordinator-t, ha viszont nem engedélyezett, akkor megállítjuk a tweet analyzer coordinator-t. Ezután a szervíz folyamata újraindul mindkét esetben.

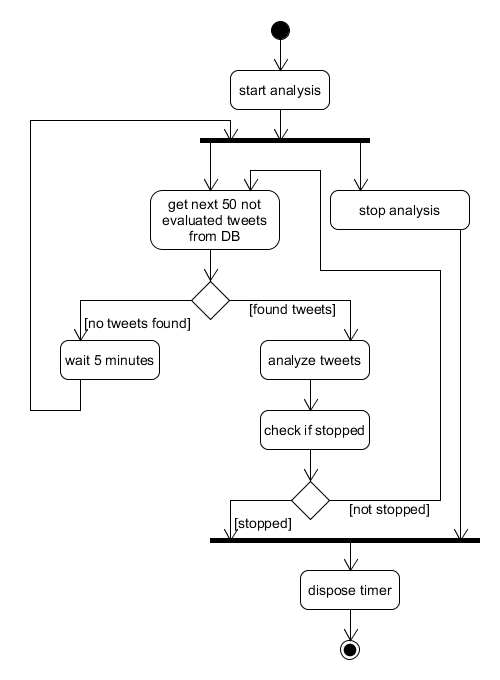


**TweetAnalyzerCoordinator** osztály szerepe a tweetek érzelem elemzési folyamatának a vezérlése. A \_logger adattag segít a folyamatok naplózásában. A \_tDao adattag segítségével hajtunk végre majd lekérdezést az adatbázis tweet táblájából. A \_timer adattaggal időzítjük a tweet érzelem elemző folyamat újraindítását. Az \_isStopped adattag mondja meg, hogy az elemzés fut vagy meg van állítva.

Az osztály konstruktorában beállítjuk a \_logger adattagnak a kapott paraméter értékét és inicializáljuk a \_tDao adattagot. A StartAnalyzingTweets metodús naplózza az érzelem elemző folyamat elindítását, hamisra állítja az \_isStopped adattagot és meghívja a ContinueTweetAnalysis metodúst. A ContinueTweetAnalysis metodús előkészít egy tweetekből álló listát TryGetTweets metodús meghívásával. Ha a lista üres akkor meghívodik a StartWaitingForNewTweets metodús. Ha a listában vannak elemek, akkor inicializál egy helyi TweetAnalyzer típusú változót az analyzer névvel, majd egy ciklust futtat amíg az érzelem elemzés engedélyezve van és a lista nem üres. A cikluson belül naplózzuk, hogy érzelem elemzést hajtunk végre és hány darab tweetre. A while cikluson belül egy újabb while ciklussal megyünk végig a lista elemein míg az érzelem elemzés engedélyezve van és a lista nem üres. Ezen a második cikluson belül az analyzer-nek meghívjuk az Analyze metodúsat, amely lementi a kiértékelt eredmányeket és visszatéríti a sikersen kiértékelt tweetek számát. Miután végigmentünk a lista elemein naplózzuk az érzelem elemzés eredményét. Ezután ha az érzelem elemző folyamat engedélyezve van újratöltsük az elemzésre szánt tweetek listáját, a következő tweetekkel. Egy helyi változó segítségével követjük azt, hogy hány tweetet elemeztünk. Ha a lista elemei elfogynak meghívodik a StartWaitingForNewTweets metodús, más esetben naplózzuk, hogy megállt az elemzés. A TryGetTweets metodús paraméterként megkapja, hogy hány tweetet kell átugorjon, egy try-catch-ben meghívja a \_tDao adattagról a GetTweetsForEvaluation metodúst. Hiba esetén naplózza a hibát és null értéket térít vissza, sikeres kapcsolodás után visszatérít egy tweetekből álló listát. A StartWaitingForNewTweets metodús naplózza, hogy tweet hiány miatt várakozás következett be, majd időzíti a várakozást 5 percre. A timer\_Elapsed metodús meghívja a DestroyTimer metodúst majd utána a ContinueTweetAnalysis metodúst. A StopAnalyzingTweets metodús meghívja DestroyTimer metodúst, beállítja az \_isStopped adattagot igazra és naplózza, hogy az érzelem elemzési folyamat megállt. A DestroyTimer metodús ha a \_timer adattag nem null, akkor meghívja a rajta a Stop metodúst.

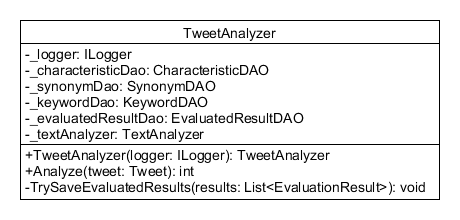


A TweetAnalyzerCoordinator folyamat azzal kezdődik hogy lekérdezzük az adatbázisból a soron következő 50 darab tweetet. Ha nincsenek tweetek akkor 5 perces várakozás után újból lekérdezzük az adatbázisról a tweeteket. Ha megvan a tweet listánk akkor elkezdjük az elemzést, utána pedig ellenőrizzük ha megállt-e az elemzési folyamat. Ha megállt a folyamat akkor megállitjuk az időzítöt és az elemzési folyamatot. Ha nincs megállítás akkor újból lekérdezzük a következő 50 darab tweetet.

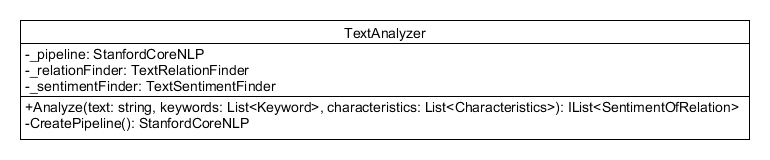


A **TweetAnalyzer** osztály szerepa a tweetek szövegének az érzelmi elemzése és az elemzés eredményeinek elmentése az adatbázisban. A \_logger adattag segít a folyamatok naplózásában. A \_characteristicDAO, \_synonymDAO, \_keywordDAO és \_evaluatedResultDAO adattagokon keresztül érjük el az adatbázisból a Characteristic, Synonym, Keyword és EvaluatedResult táblákat. A \_textAnalyzer adattag segít a tweet szövegének az érzelem elemzésére való előkészítésében.

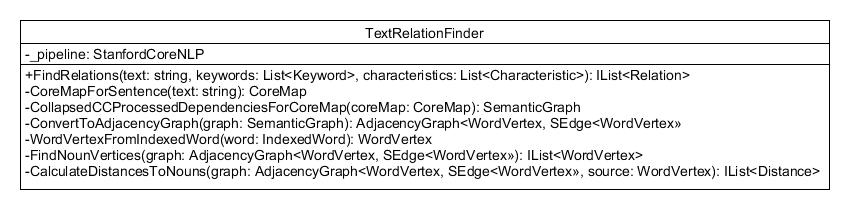
Az osztály konstruktorában a \_logger adattagnak átadjuk a paraméter értékét, valamint inicializáljuk a \_characteristicDAO, \_synonymDAO, \_keywordDAO, \_evaluatedResultDAO és \_textAnalyzer adattagokat. Az Analyze metodús paraméterként megkapja azt a tweetet amelynek a szövegjén az érzelmi elemzést fogjuk végezni. Deklarálunk egy lokális változót amely a talált érzelmek számát fogja nyilvántartani, totalSentiments néven kezdetben zéros értékkel. Egy TextPreprocessor típusú lokális változót is inicalizálunk preprocesser néven. Lekérdezzük a categoryIds lokális változóba a tweetben szereplő összes kategoriát. A categoryIds lista minden elemére lekérdezzük az adott kategoriához tartozó kulcsszavakat, szinonimákat és tulajdonságokat, a keywords, synonyms és characteristics lokális listákba. A keywords és characteristics lista elemeiből elkészítsük a keywordsForAnalyzer és characteristicsForAnalyzer lokális listákat. A preprocessor lokális változón meghívjuk az Preprocess metodúst átadva paraméterként a tweet szövegét, a szinonimák és a kulcsszavak listáját és visszakapunk egy preprocessedText lokális string típusú változót. Meghívjuk a \_textAnalyzer adattag Analyze metodúsát átadva paraméterként a preprocessedText változót, a keywordsForAnalyzer és characteristicsForAnalyzer listákat, visszakapunk egy sentiments lokális listát, amely IList<AM.SentimentOfRelations> típusú. Inicializáljuk a results listát amely EvaluationResult típusú. Foreach ciklussal bejárjuk a sentiments lista elemeit. A results listát feltöltjük a sentiments lista elemeiben eltárolt eredményekkel. Meghívjuk a TrySaveEvaluatedResult metodúst ha a results lista nem üres. Frissitjük a totalSentiments változó értékét az újonnan sikeresen lementett érzelmi eredmények számával, majd visszatérítjük az értékét. A TrySaveEvaluatedResult metodús paraméterként megkapja az EvaluationResult típusú listát. Try-catch-ben naplózza az információt arról, hogy hány érzelmi elemzést fog lememteni az adatbázisba és meghívja az \_evaluatedResultDao adattagról a SaveEvaluatedTweets metodúst továbbadva paraméterként az eredmények listáját. Hiba fellépése esetén naplózza az információt, hogy az adatbázisba nem sikerült lementeni az adatokat.



A **TextAnalyzer** osztály szerepe a tweet szövegének a Stanford Parser segítségével a mondattani és érzelmi kielemzése. A \_pipeline adattag biztosítja a hozzáférést a Stanford Parser szövegelemzőjéhez. A \_relationFinder és a \_sentimentFinder adattagok segítségével fogjuk megkeresni a szövegben a kulcsszavak és tulajdonságok közti kapcsolatokat, valamint az érzelmeket megkeresni az előbb megtalált kapcsolatokban. Az Analyze metodús paraméterként megkapja az elemzésre szánt szöveget, a kulcsszavak és tulajdonságok listáját. A \_relationFinder adattagon meghívja a FindRelations metodúst átadva a szöveget, a kulcsszavakat és a tulajdonságokat visszakapva a relations Relation típusú listát. A result SentimentOfRelation típusú listát feltöltjük a \_sentimentFinder adattagon a FindSentiments metodús meghívásával, amelynek paraméterként átadjuk a szöveget és relations listát. A metodús eredményként visszatéríti a result listát. A CreatePipeline metodús inicializálja és visszatéríti a Stanford Parser szövegelemzőt. Egy java.Util.properties típusú változó segítségével megadjuk az általunk felhasznált annotátorokat mint: “tokenize, ssplit, pos, lemma, parse, sentiment”, valamint a Stanford Parser által használt modelek útvonalát.



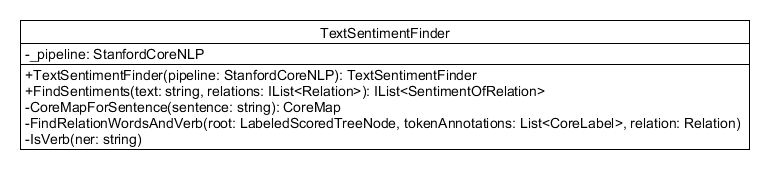
A **TextRelationFinder** osztály szerepe a tweet szövegében megkeresni a kapcsolatokat a kulcsszavak és tulajdonságok között. A \_pipeline adattag biztosítja a hozzáférést a Stanford Parser szövegelemzőjéhez. A FindRelations metodús paraméterként megkapja az elemezendő szöveget, a kulcsszavak és a tulajdonságok listáját, végül visszatérít egy Relation típusú elemekből álló listát. Először meghívja a CoreMapForSentence metodúst átadva paraméterként a szöveget és a visszatérített értéket tárolja a coreMap CoreMap típusú változóban. Ha ez a változó null, akkor a metodús visszatérít egy üres listát. Ha viszont van értéke, akkor meghívja a CollapsedCCProcessedDependenciesForCoreMap metodúst átadva a coreMap változót mint paraméter. A meghívott metodús eredményét a dependencies SemanticGraph típusú változóba tároljuk. A SemanticGraph típusú változót átadjuk mint paraméter a ConvertToAdjacencyGraph metodúsnak a visszatérített értéket pedig a graph AdjacencyGraph típusú változóba tároljuk. Inicializáljuk a distancesByNoun Dictionary<WordVertex, List<Distance>> típusú változót. Meghívjuk a FindNounVertices metodúst, átadva a graph változót paraméterként, a visszakapott WordVertex típusú elemkből álló listát eltároljuk a nounVertices változóba. Ez a lista tartalmazza az összes főnév típusú csúcspontot. Végighaladva a nounVertices lista elemein, kiszámoljuk a CalculateDistancesToNouns metódus segítségével a távolságot az éppen aktuális főnévtől a graph-ban szereplő összes többi főnévhez. Az így visszatérített listát eltároljuk a distancesByNoun szótárban az éppen aktuális főnév értéket használva, mint kulcs. Létrehozzuk a filtered nevű, List<Distance> típusú listát, melybe azokat a távolságokat fogjuk tárolni, melyek kulcsszó és tulajdonság közötti kapcsolatot jelölnek. Végigjárjuk a distancesByNoun szótár azon kulcs értékeit, melyek kulcsszavak. Meghatározzuk a kulcsszóhoz legközelebb eső tulajdonságokat. Ha e tulajdonságokhoz nem létezik más, közelebb eső kulcsszó, mint az éppen aktuális, akkor betesszük a filtered nevű listába. Ha e tulajdonságokhoz létezik ugyanolyan távolságra eső más tulajdonság, mint a kulcsszó és tulajdonság közti távolság, akkor betesszük a filtered listába a kulcsszó és a másik tulajdonság közti távolságot is. A metódus végén végigjárjuk a filtered listát és átalakítjuk az elemeket Relation típusú objektumokká, melyeket a result List<Relation> típusú listába tárolunk. Végül ezt a listát térítjük vissza a metódusból. A CoreMapForSentence metodús paraméterként megkapja az elemezendő szöveget és visszatéríti a Stanford Parser által annotált mondatot, az annotációk típusát a \_pipeline adattag tartalmazza. A CollapsedCCProcessedDependenciesForCoreMap metodús paramáterkánt megkapja a coreMap változót és ez alapján ugyancsak a Stanford Parser-t használva visszatérít egy SemanticGraphCoreAnnotations.CollapsedCCProcessedDependenciesAnnotation típusú szematikus gráfot, abban az esetben ha a coreMap paraméternek van értéke. A ConvertToAdjacencyGraph metodús megkap egy SematicGraph típusú paramétert és ezt átalakítja AdjacencyGraph<WordVertex, SEdge<WordVertex>> típusú gráffá. A WordVertexFromIndexedWord metodús az IndexedWord típusú paramétert alakítja át saját WordVertex típusú csúcsnak. A FindNounVertices metodús a AdjacencyGraph típusú paramáterből előkészít egy WordVertex típusú elemekből álló listát ami tartalmazza a gráfban szereplő összes főnevet. A CalculateDistancesToNouns metodúsnak átadjuk paraméterként egy AdjacencyGraph típusú gráfot és egy WordVertex típusú csúcsot. A metodús felhasználva a QuickGraph-nak a ShortestPathsDijkstra metodúsát felépít egy Distance típusú elemekből álló listát, ami tartalmazza a mondatban szereplő összes főnevek közti legkisebb távolságokat.



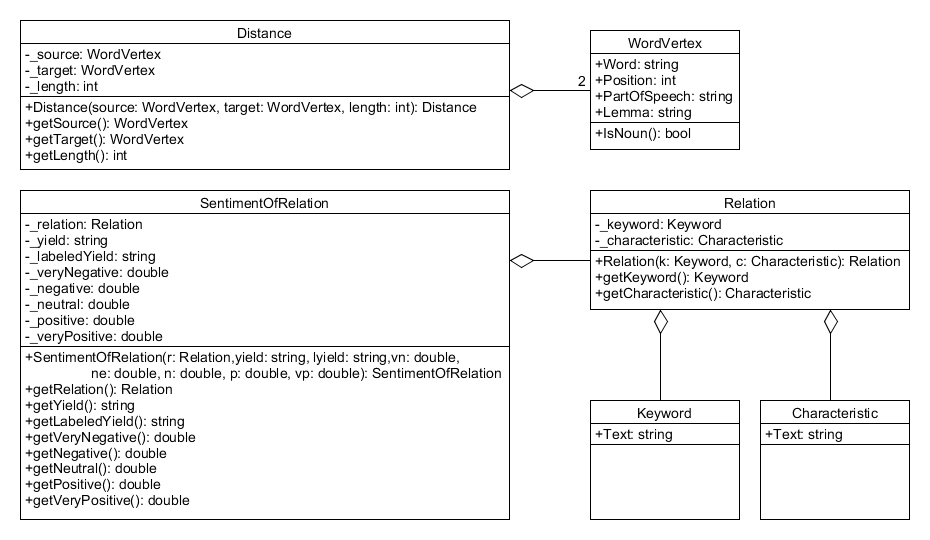
A **TextSentimentFinder** osztály szerepe az érzelmi elemzés elvégzése. A \_pipeline adattag biztosítja a hozzáférést a Stanford Parser szövegelemzőjéhez. Az osztály konstruktorában a \_pipeline adattagnak átadjuk a kapott paraméter értékét. A FindSentiments metodús paraméterként megkapja az elemzésre szánt szöveget és egy Relation típusú elemkből álló listát. Amennyiben a paraméterként kapott lista üres, visszatérítünk egy üres SentimentOfRelation elemekből álló listát. Felhasználva a CoreMapForSentence metódust, átalakítjuk a paraméterként kapott szöveget egy CoreMap típusú objektummá. A CoreMap típusú objektumból kiszűrünk egy olyan CoreLabel típusú elemeket tartalmazó listát, melyek el vannak látva TokensAnnotation típusú annotációval. Szintén a CoreMap objektumból kivesszük a SentimentCoreAnnotation-el ellátott érzelmi fát. A továbbiakban végighaladunk a paraméterként kapott relációkon, és minden relációra meghatározzuk a reláció érzelmi értékét FindRelationWordsAndVerb metódus segítségével, átadva a CoreLabel listát, az érzelmi fát és a relációt. A metódus végén ellenőrizzük, hogy ugyanannyi érzelmi értéket találtunk, mint ahány relációt kaptunk paraméterként. Amennyiben az érzelmi értékek száma kisebb, mint a relációk száma kivételt dobunk. A metódus végén visszatérítjük az érzelmi értékeket tartalmazó listát. A CoreMapForSentence metodús paraméterként megkapja az elemezendő szöveget és visszatéríti a Stanford Parser által annotált mondatot, az annotációk típusát a \_pipeline adattag tartalmazza.

A FindRelationWordsAndVerb metodús segítségével határozzuk meg az érzelmi értékét egy relációnak. Ehhez felhasználjuk a paraméterként kapott érzelmi fát és TokensAnnotation-al ellátott CoreLabel listát. A metódus úgy működik, hogy végigjárja az érzelmi fát poszt-order bejárást használva. Az érzelmi fa csomópontjai szövegként tartalmaznak egy-egy mondatrészt, amint ezt a 3.5 fejezetben már ismertettem. A metódus addig halad a fa csomópontjain, míg egy olyan csomópontot nem talál, melyben szerepel a kulcsszó, a tulajdonság és egy ige. Ennek a csomópontnak az érzelmi értékét téríti vissza eredményként.

Az IsVerb metodús a kapott string típusú paraméterre megnézi, hogy igeként van-e annotálva a Stanford Parser által vagy nem.

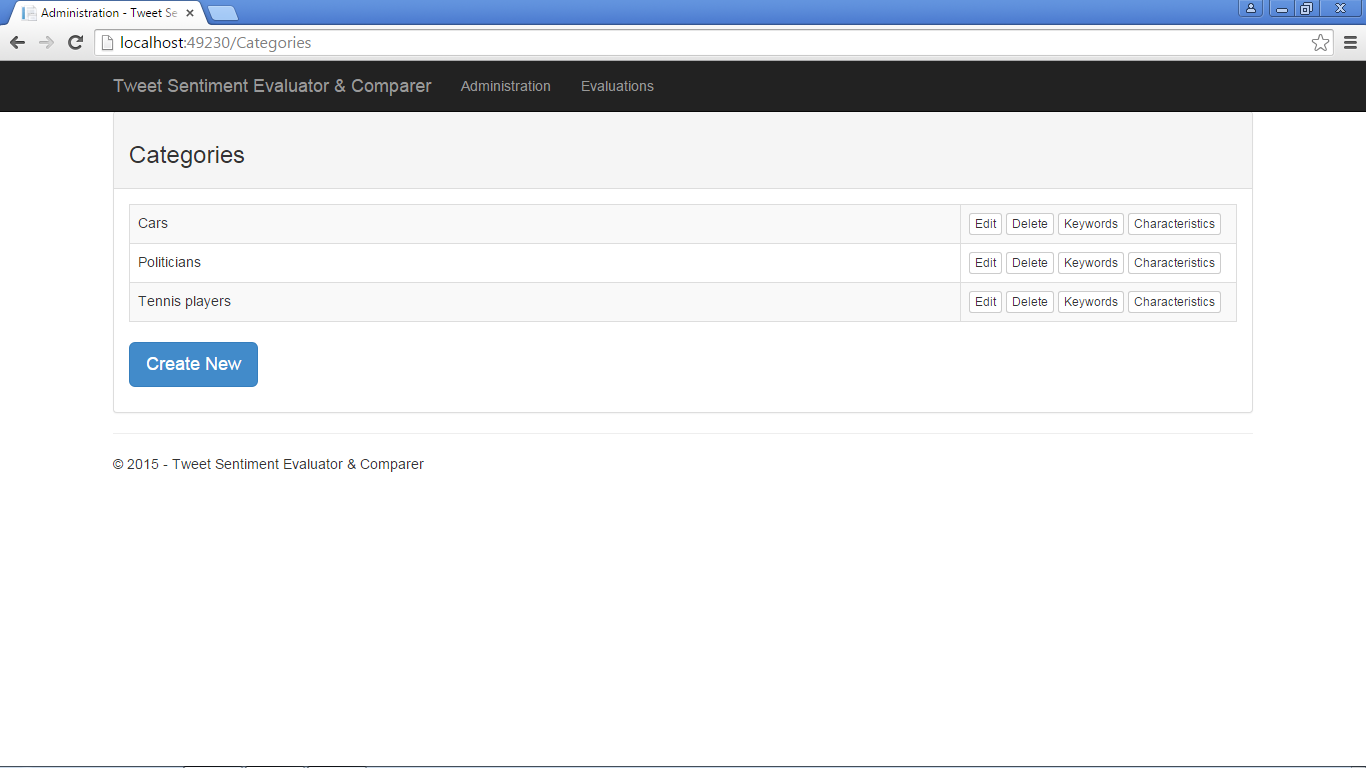


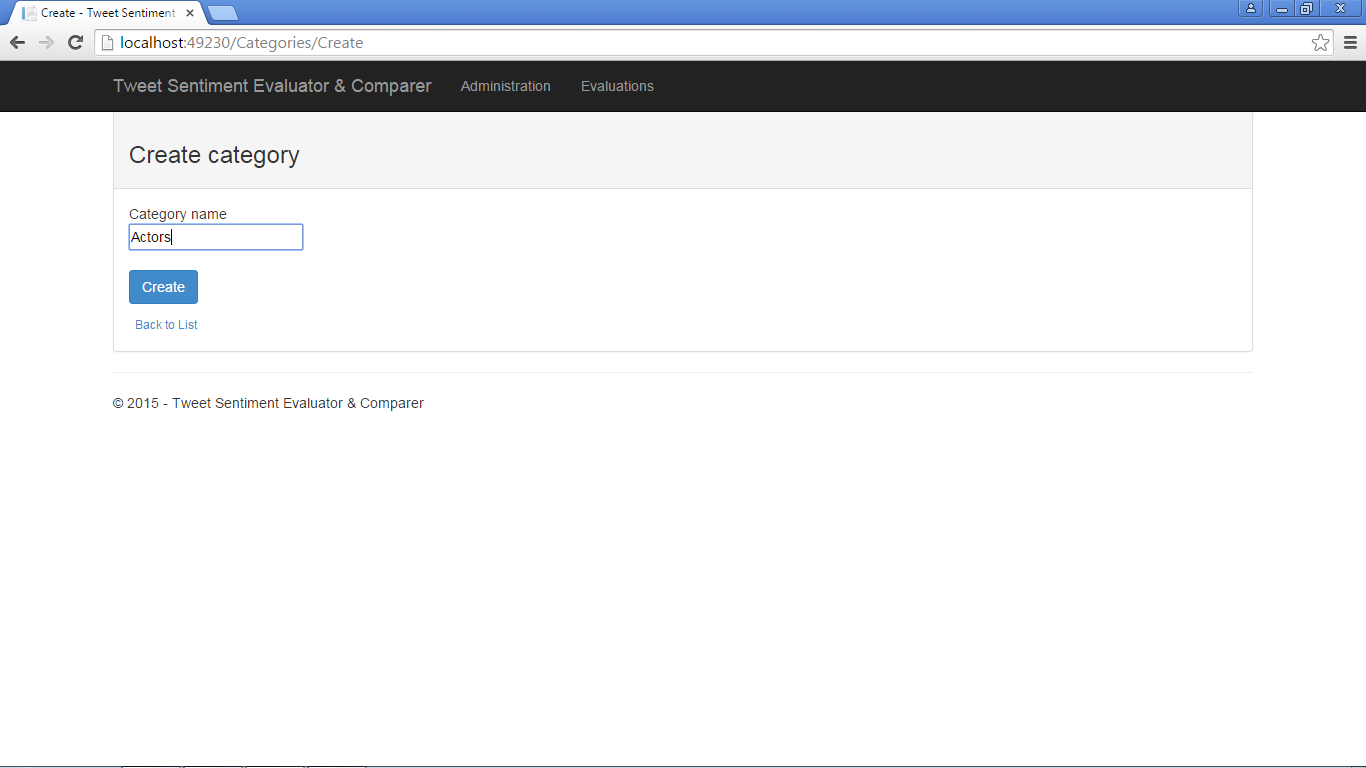
A segéd modell osztályok azért voltak létrehozva, hogy az érzelemelemző modul osztályai ezek segítségével működjenek együtt. A Distance és a WordVertex osztályok a QuickGraph típusú gráfokban levő távolságokat, valamint szó-csomópontokat reprezentálják. A távolságnak van egy forrása és célja, valamint egy értéke. A szó-csomópont eltárolja a szót, mint karakterlánc, a mondatbeli pozícióját, hogy milyen szótani szerepet tölt be, valamint a szó lemmatizált alakját. A Keyword és Characteristic osztályok egy-egy karakterláncot tárolnak. A Relation osztály egy kulcsszó és karakterisztika közti relációt modellez. A SentimentOfRelation osztály egy relációt tartalmaz, 2 mondatrészletet és az érzelmi értékeket százalékosan kifejezve. A yield mondatrészlet, a mondat azon részét tartalmazza, melyben szerepel a reláció kulcsszava, tulajdonsága és egy ige. A labeledYield mondatrészlet ugyanazokat a szavakat tartalmazza, mint a yield, csak minden szó után szerepel a szófaja pl: „Engine/NN is/VBZ great/JJ”.

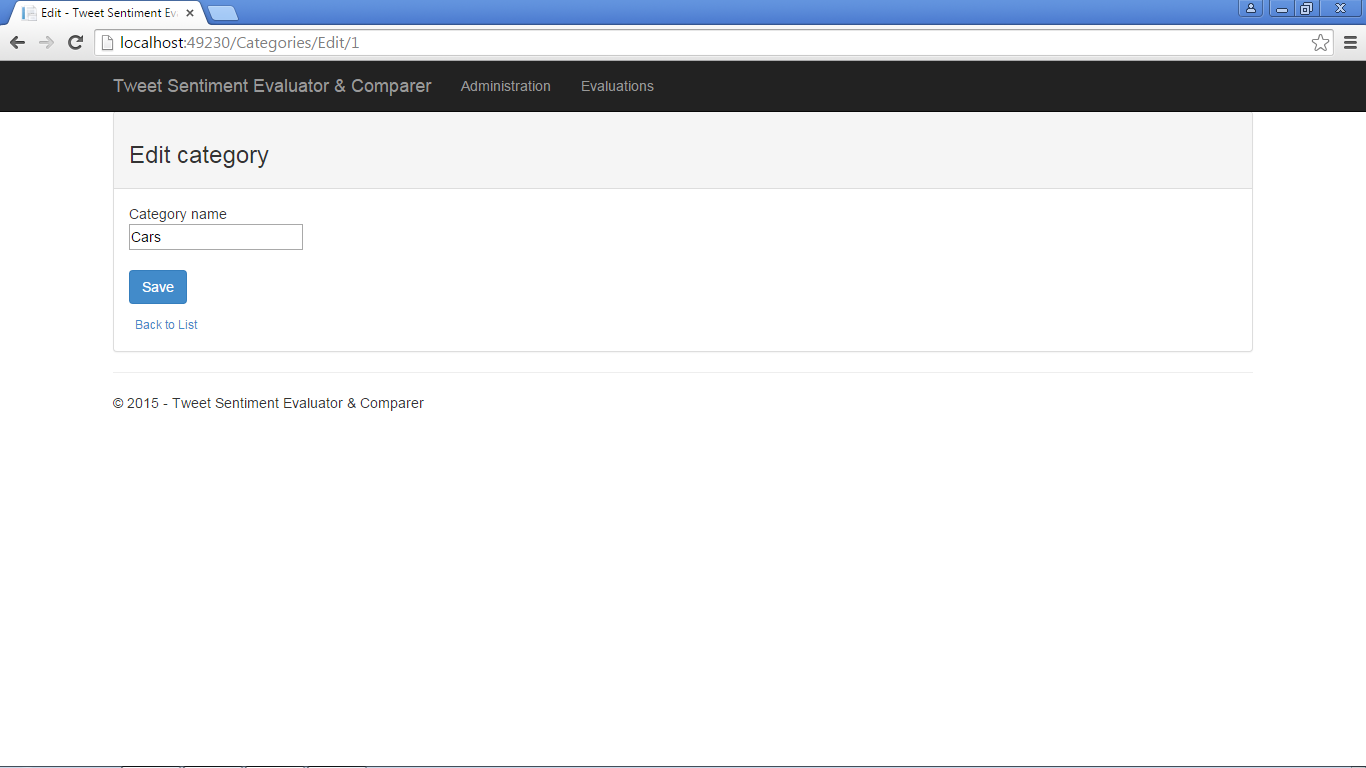


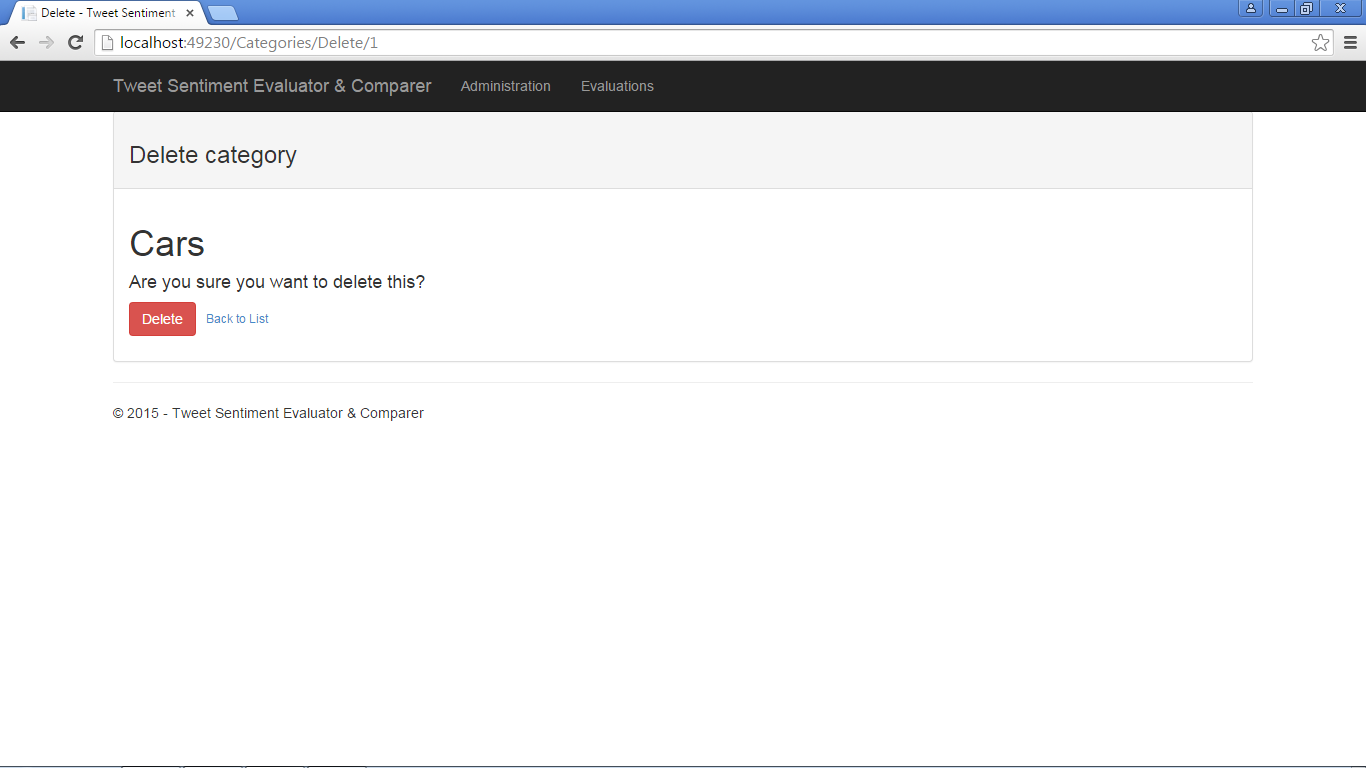
### Adminisztrációs modul

Az adminisztrációs modul web applikáción keresztül valósítottam meg, a megjelenítéshez használtam a Twitter Bootstrap által kinált lehetőségeket. Ezen a modulon keresztül a felhasználó megjelenítheti listák formájában az általa használt kategoriákat, kulcs-szavakat, szinonimákat és tulajdonságokat. A kategoriák listába vihet be új elemeket, modosíthat és törölhet létező elemeket. A kulcs-szavak, szinonimák és tulajdonságok listájába vihet be új elemeket és törölhet létezöket.





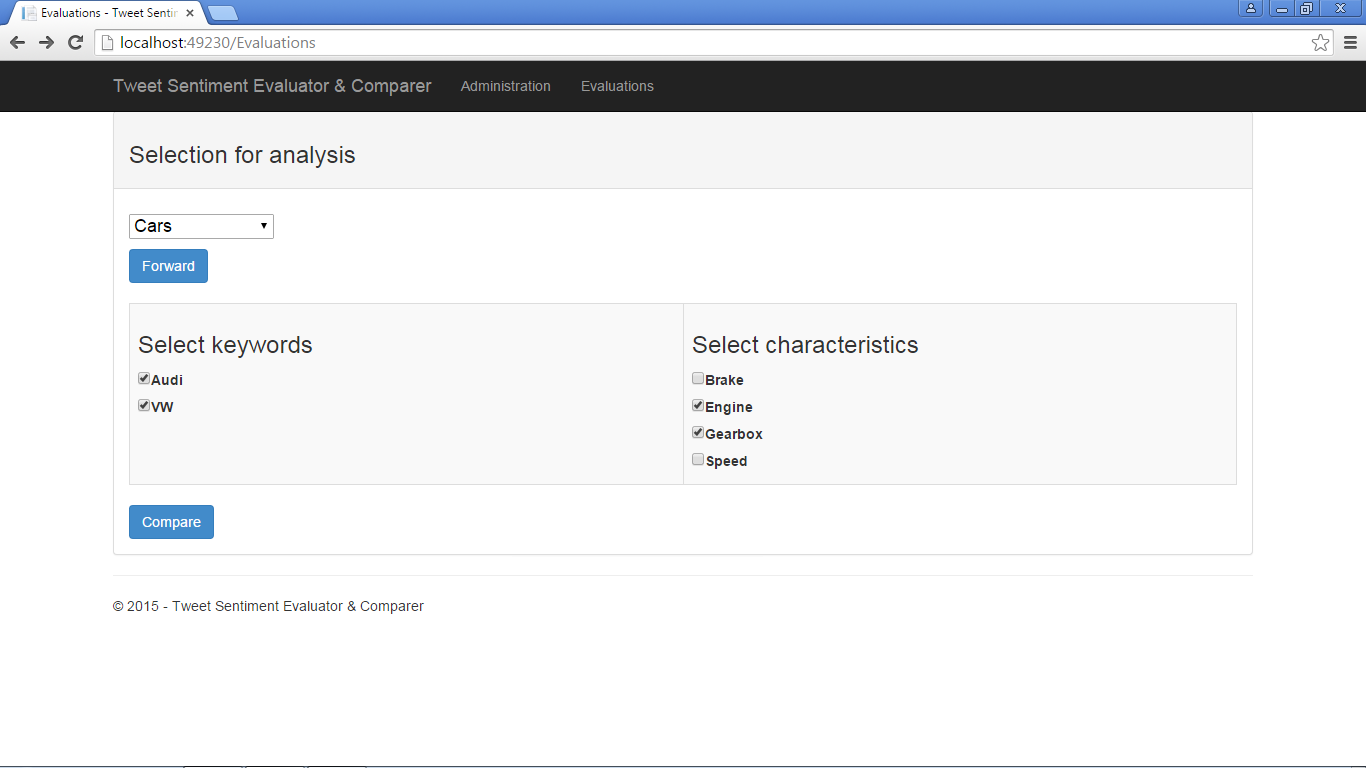


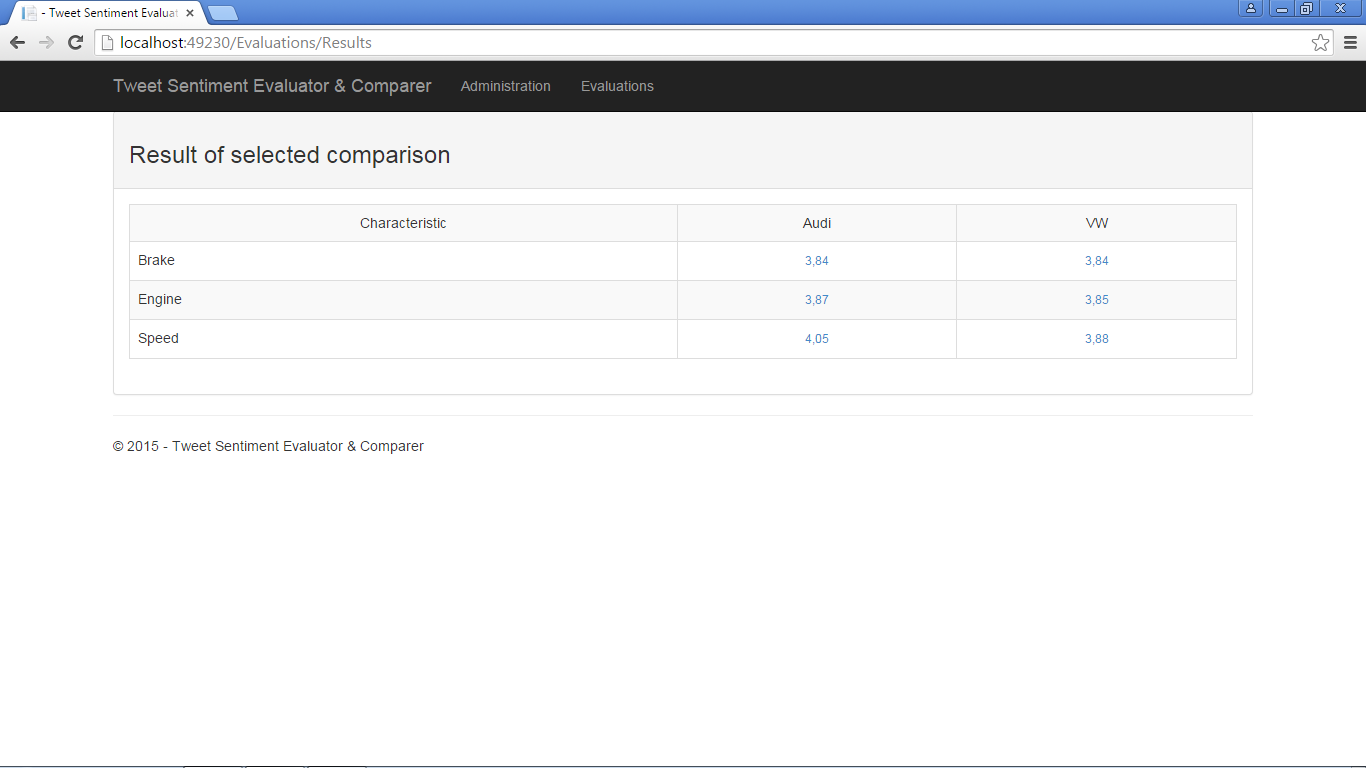


### Eredményjelző modul

Az eredményjelző modul a web aplikáció azon része mely a felhasználó számara megjeleníti az érzelemelemző modul által kiértékelt eredményeket. Két fő részre osztható a következő szerepekkel: a kiértékelési feltételek beállítása és az eredmények megjelenítése. A feltételek meghatározásánál a felhasználó elsősorban kiválaszt egy kategoriát, majd a kategorián belül létező kulcsszavak és tulajdonságok közül választhat több elemet is az összehasonlítás elvégzéséhez.

Az összehasonlítás eredménye egy táblázatként jelenik meg. Az elért kiértékelésre kattintva az applikáció egy ablakban feltünteti az adott kulcs-érték, tulajdonság párosra vonatkozó első öt tweet szövegét.







## A rendszer felhasználása

A rendszer felhasználása web aplikációs felületen keresztül történik. A web felület három fő részre osztható:

* adminisztrációs
* szervízek üzemeltetése
* eredmények vizsgálata

Az adminisztrációs felületen a felhasználó megtekintheti és modósíthatja a rendszer által használt kategóriákat, az adott kategoriákon belül a kulcsszavakat és tulajdonságokat, a kulcsszavakon belül a szinonimákat.

A szervízek üzemeltetésének a beállításával a felhasználó elindíthatja vagy megállíthatja a két szervízt, a tweetek gyűjtését és az érzelem elemzést. A tweet gyűjtési szervíz engedélyezésével elindul a tweetek keresése és letöltése az internetről. Az aplikáció a felhasználó által definiált szinonimákat fogja keresni a tweetek szövegében és a találatokat lementi egy saját adatbázisba. Az érzelem elemzési szervíz engedélyezésével elindul a letöltött és még ki nem elemzett tweetek szövegének az érzelmi kiiértékelése.

Az eredmények megtekintéséhez a felhasználó kiválaszthatja a kívánt kategóriát, ezen belül választhat a kulcsszavak és tulajdonságok közül. A rendszer egy összehasonlító táblázatban mutatja, a kiértékelt eredményeket, a kiválasztott kritériumoknak megfelelően. A felhasználó egy adott eredményre, amely egy kulcsszó-tulajdonság páros eredménye, megtekintheti pár tweetnek a szöveget valamint az érzelem eredmény alapjául szolgáló véleményeknek a skáláját.

## Üzembehelyezés és kísérleti eredmények

### Felhasznált technológiák

A következőkben a rendszerben használt technológiákat mutatom be. A rendszer webalkalmazás komponense a Microsoft által létrehozott ASP.NET MVC 4-es verziójú technológiára épül. A webalkalmazás felhasználja a Twitter Bootstrap 3.3.4-es verzióját szebb megjelenítés elérésének az érdekében. A rendszer az adatbázissal való kapcsolatot az Entity Framework 6-os verziójú technológia felhasználásával valósitja meg. A rendszer két fontos komponense, a tweet letöltés és az érzelem elemzés, ami windows szervízként futtatható, a ServiceProcess 4-es számú verziójú technológiát használja fel. A tweetek letöltését biztosító komponens a 0.9.6.1-es verziójú Tweetinvi. A szövegfeldolgozását és érzelmi kiértékelését szolgáló érzelmi elemző komponens a 3.5.1-es verziójú Stanford Core NLP technológiára épül. Úgyanez a komponens még használja a 3.6.61114.0-ás verziójú Quickgraph technológiát is. Az adatok tárolására használt adatbázis a 12.0.2000.8-as verziójú Microsoft SQL Server 2014.

### Felmerült problémák és megoldásai

A következőkben leírom, hogy a rendszer megvalosítása során miyen problémákba ütköztem, és hogyan sikerült ezeket megoldanom vagy kiküszöbölnöm.

Az első problémát a Twitter API tweet letöltési korlátozásai okozták. A Twitter API korlátozza a lekérdezések közti időtartamot, ami azt jelenti, hogy 180 tweet letöltési HTTP kérés után, megközelítőleg 15 percet kell várakozni, míg újra folytatni lehet a letöltést. Ez azt a problémát okozta, hogy a felhasználó elindítja a letöltési folyamatot és a letöltések befejezése vagy az időkorlát elérése után megállt volna a letöltési folyamat. Ahhoz, hogy a letöltési folyamat folytatódjon, szükség lett volna egy manuális újraindításra, ami nem megengedhető egy ilyen rendszer esetében. A problémát egy időzítő használata segítségével oldottam meg. Az időzítő a letöltési folyamat kezdetén a Tweet API-tól lekérdezi az adott idő- és kéréskorlátot és követi ezeknek a lejártát. Ha elértük a letöltési kérés korlátot, a rendszer addig várakozik, míg a Twitter által megszabott időkorlát le nem jár, majd folytatja a tweet-ek keresését. Ha a letöltés befejeződik, mivel nem találunk több tweet-et egy adott szinonimának, akkor egy következő szinonimát keresünk az adatbázisból, és tovább folytatódik a keresés.

A második probléma a szövegelemzés során merült fel, a kulcsszavak és tulajdonságok keresése során. A felhasználó által keresett szavak legtöbb esetben egyes számban vannak megadva, például “wheel”, viszont az elemzett szövegben előfordulhatnak többes számban vagy különböző ragokkal ellátva, mint: wheels. A probléma megoldásához a keresések végzéséhez a szavak lemmatizált formáját használtam.

A harmadik probléma szintén a szövegelemzés során jött elő, a tweetek szövegében található linkek, hashtag-ek. Ezekre a szavakra nem igazán végzett helyes szófaji annotációt a Stanford Parser szövegelemzője. A probléma megoldására egy szöveg előfeldolgozást használtam, amely során a helyettesítem a http-vel kezdő linkeket a LINK, a @ jellel kezdödő szövegrészeket a NAME, valamint a # jellel kezdödő szövegrészeket a CATEGORY szavakkal.

A negyedik problémát a szervízek webes felületről történő elindítása jelentette. Mivel a szervízek Windows szervízként futnak nem indíthatok el webes felületről, mivel ahhoz, hogy egy Windows szervízt egy program elindítson, adminisztrátori jogokra van szüksége. Egy olyan megoldást kellett találni, mely segítségével a két komponens képes lesz egymással kommunikálni. A megoldás amit alkalmaztam egy flag bevezetése volt, amit az adatbázisban tároltam, mivel az adatbázishoz úgy a web aplikáció, mint a windows szervízek hozzá kell férjenek. Ezt a flag-et a web aplikáción keresztül módosíthatja a felhasználó. A szervíz egy előre definiált időközönként ellenőrzi az adatbázisban a flag értékét és ennek megfelelően elindítja a munkáját vagy megállítja azt.

### A relációkeresés eredményei

Azt, hogy a reláció meghatározása mennyire hatékony, unit tesztek segítségével ellenőriztem. A továbbiakban, ezekre csak tesztként fogok hivatkozni. Mint azt már az előző fejezetekben ismertettem, egy mondaton belül a relációt egy kulcsszó és egy tulajdonság között kell meghatározni. Ezt a feladatot az is bonyolíthatja, hogy a relációkereső komponens nem minden szóról ismeri, hogy az kulcsszó-e vagy tulajdonság-e. Éppen ezért a teszteket ennek megfelelően alakítottam ki, így ezek tartalmaznak ismert és valódi kulcsszavakat (tulajdonságokat). Ismert szó alatt azt értem, hogy a rendszerben szerepel az adott szó, mint kulcsszó, vagy mint tulajdonság. Valódi szó alatt, pedig azt, hogy a rendszerben nem szerepel a szó, mint kulcsszó, vagy mint tulajdonság. A következő táblázat összefoglalja a különböző eseteket, melyeket vizsgálnak a tesztek (IK – Ismert kulcsszavak, IT – Ismert tulajdonságok, VK – Valódi kulcsszavak, VT – Valódi tulajdonságok):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IK** | **IT** | **VK** | **VT** | **Mondat** |
| 2 | 2 | 2 | 2 | The engine of Audi is great, but VW has the worst brakes. |
| 2 | 2 | 1 | 2 | The brake and engine of this Audi is great. |
| 2 | 2 | 2 | 1 | The brake of this Audi and VW is great. |
| 2 | 2 | 2 | 2 | Audi and VW have good engines and brakes. |
| 2 | 2 | 2 | 1 | Audi has better brakes than VW. |
| 2 | 2 | 2 | 2 | Audi has a powerful engine, while VW has the best brakes. |
| 2 | 1 | 2 | 2 | The engine of Audi is great, but VW has the worst brakes. |
| 2 | 1 | 1 | 2 | The brake and engine of this Audi is great. |
| 2 | 1 | 2 | 1 | The brake of this Audi and VW is great. |
| 2 | 1 | 2 | 2 | Audi and VW have good engines and brakes. |
| 2 | 1 | 2 | 1 | Audi has better brakes than VW. |
| 2 | 1 | 2 | 2 | Audi has a powerful engine, while VW has the best brakes. |
| 1 | 2 | 2 | 2 | The engine of Audi is great, but VW has the worst brakes. |
| 1 | 2 | 1 | 2 | The brake and engine of this Audi is great. |
| 1 | 2 | 2 | 1 | The brake of this Audi and VW is great. |
| 1 | 2 | 2 | 2 | Audi and VW have good engines and brakes. |
| 1 | 2 | 2 | 1 | Audi has better brakes than VW. |
| 1 | 2 | 2 | 2 | Audi has a powerful engine, while VW has the best brakes. |
| 1 | 1 | 2 | 2 | The engine of Audi is great, but VW has the worst brakes. |
| 1 | 1 | 1 | 2 | The brake and engine of this Audi is great. |
| 1 | 1 | 2 | 1 | The brake of this Audi and VW is great. |
| 1 | 1 | 2 | 2 | Audi and VW have good engines and brakes. |
| 1 | 1 | 2 | 1 | Audi has better brakes than VW. |
| 1 | 1 | 2 | 2 | Audi has a powerful engine, while VW has the best brakes. |

A táblázatban 24 eset szerepel, ellenben összesen 45 teszt létezik, mivel olyan esetekben, amikor a relációkereső komponens nem ismer csak 1 kulcsszót (vagy tulajdonságot) és a mondatban 2 valódi kulcsszó (vagy tulajdonság) szerepel, 2 tesztet is írtam. Az egyik tesztben az első kulcsszót (vagy tulajdonságot), a másik tesztben a második kulcsszót (vagy tulajdonságot) jelöltem be, mint ismert kulcsszó (vagy tulajdonság). Ahhoz, hogy egy tesztesetet sikeresnek nyilvánítsunk, a relációkereső komponens a következő feltételeknek kell, eleget tegyen:

* találja meg a valódi relációkat
* ne találjon olyan relációkat, melyek nem léteznek
* csak a valódi relációkat találja meg

A 45 teszt esetében a relációkereső komponens 39 esetben helyesen állapította meg a mondatban szereplő kulcsszó-tulajdonság relációt, 6 esetben pedig tévesen. Ez százalékosan kifejezve 86.66%-os helyes meghatározást jelent.

Pie chart….

A relációkereső olyan esetekben téved, amikor a mondatban 2 kulcsszó és 2 tulajdonság szerepel, és a kereső ismeri az egyik kulcsszavat és azt a tulajdonságot, mely a másik kulcsszóval áll relációban. Egy példa erre az esetre: „The engine of Audi is great, but VW has the worst brakes.”, ismert kulcsszó: „Audi”, ismert tulajdonság: „brake”. A relációfelismerő felismeri, mint reláció az „Audi” – „brake” relációt, holott ez a kulcsszó-névelem páros nincs relációban.

* Táblázatok, grafikonok
* Használhatóak-e az eredmények

# Következtetések

## Megvalósítások

what is the system useful for

what did YOU learn from this project

A rendszer egy több komponensből álló aplikáció, melynek a felhasználói felülete egy webalkalmazás ami C# nyelvben írodott a Microsoft Visual Studio Community 2013-as környezetben. A rendszer tartalmaz két Windows Service-ként futtatható komponenst. A rendszer az adatokat az SQL SERVER adatbázisba perszisztálja.

## Hasonló rendszerekkel való összehasonlítás

Sentimentor – egy olyan eszköz, amely a Twitter adatok érzelem elemzését végzi. [12] Ez az eszköz a a naív Bayes Classifier-t használja arra, hogy a tweeteket pozítiv, negatív és objektiv osztálykba sorolja. Az eszköz az elemzés előtt első lépésként eltávolítja az összes linket és felhasználónevet a tweetek szövegéből, valamint az olyan szavakat amelyeknek önmagukban nincs jelentésük, mint például kötőszavak, határozószavak. A második lépésben szófaji annotációval látja el a szöveget valamint előállítja a szövegből az unigramok és bigramok listáját, használva az OpenNLP könyvtárat. A Sentimentor egy web-aplikáció amely a szavak előfordulásáról készít kimutatásokat, érzelem elemzést végez szöveg-részre, keresést hajt végre a Tweetek között.

Az Umigon aplikáció szókincs és heurisztikára alapozott érzelem elemzést végez a tweeteken. [13] Speciálisan arra volt tervezve, hogy a tweetekben a pozítiv, negatív vagy semleges érzelmeket felfedezze. Az Umigon érzelem felfedező motorja négy fő részből áll:

* Szemantikus jellemvonások felfedezése a tweetekben, külön odafigyelve a smiley-ra
* Hashtag-ek kiértékelése
* n-gramok listájának előállítása, legfeljebb 4-gram-ig, az n-grammokat összehasonlítja a már létező lexikonnal és találat esetén kiértékelést végez rá
* felhasználva az elöző lépések során felfedezett szemantikus jellemvonásokat, elvégzi a végleges kiértékelést a teljes tweetre és egyetlen érzelmet csatol a tweetnek

Az általam készített aplikáció bizonyos szempontókból hasonlít a megemlített aplikációkra és vannak olyan részek amelyekben különbözik.

Hasonló részek:

* A szöveg előkészítése érzelem elemzésre
* Szófaji annotációk elvégzése a szavakra

Különböző részek:

* Az érzelem kiértékelése nem a teljes tweetre hajtódik végre
* Nem használok n-gramokat, hanem saját módszerrel keresem meg az érdekelt szövegrészt
* Az általam készített érzelem elemző fő sajátossága abban van, hogy nem a teljes mondatra végez érzelem kiértékelést, hanem a kulcsszó – tulajdonság reláció megkeresése után előállított szövegrészre.

## További fejlesztési irányok

### Tweetek gyűjtése Stream API-val

Az aplikáció tweet gyűjtő moduljának a bővítése a Tweet API által biztosított Streaming API használásával. Ily modon hozzáférhetünk a tweetekhez abban a pillanatban ahogy létrejönnek, sőt mivel a Search API lekérdezésekkel nem elérhető az összes létező tweet, így biztosak lehetünk abban, hogy nem szalasztunk el tweeteket. Természetesen ezt a Stream API-t felhasználó letöltési modult ahhoz, hogy ne szalasszon el tweeteket állandóan üzemeltetni kéne.

### Tweet szöveg újra - kiértékelése

Egy másik tovább fejlesztési lehetőség az lenne, hogy azokat a tweeteket amiket már kiértékeltünk az érzelem elemző modullal egy adott pillanatban újra kiértékeljük. Ez azért lenne hasznos hisz ha bővülnek a keresett kulcsszavak vagy tulajdonságok, a kiértékelés pedig csak azokra a kulcsszavakra és tulajdonságokra történt amik a kiértékelés időpontjában léteztek, az előzöleg kiértékelt tweetek tartalmazhatnak információt az újonnan bevitt kulcsszavakról vagy tulajdonságokról is. Ennek a célnak az elérése érdekében használhatnánk egy hash függvényt (pl. SHA, MD5), ami segítségével azonosítanánk az adott tweet melyik kulcsszavak és tulajdonságokra volt kiértékelve.

### Adatforrások bővítése

Az aplikáció csak a Twitteren megjelent tweeteket használja adatforrásául, de szerintem sokkal bővebb és talán szakmaibb információkat is találhatunk a sajátos úgynevezett review-s web oldalakon vagy különböző fórumokon. Itt viszont lehetséges, hogy az adatok eléréséhez szükséges lesz a web oldal tulajdonosának a beleeggyezése.

### Rendszerállapot kijelzés

A felhasználó részére kimutatások készítése olyan információkkal mint a kulcsszavanként hány darab tweetet töltött le a rendszer, a tweetek közül hányan végzett érzelmi kiértékelést a rendszer.

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | D. Boyd és K. Crawford, „Critical Questions for Big Data: Provocations for a Cultural, Technological, and Scholarly Phenomenon,” *Information, Communication, & Society,* pp. 662-679, 2012. |
| [2] | J. S. Ward és A. Barker, „Undefined By Data: A Survey of Big Data Definitions,” *School of Computer Science University of St.Andrews, UK,* 2013. |
| [3] | W. G. Mangold és D. J. Faulds, „Social media: The new hybrid element of the promotion mix,” *Kelley School of Business, Indiana University ,* 2009. |
| [4] | B. Pang és L. Lee, „Opinion Mining and Sentiment Analysis,” *Foundations and Trends in Information Retrieval,* 2008. |
| [5] | M. Z. Arbi, „Natural Language Processing,” *Mes recherches personnelles,* 2015. |
| [6] | S. Monika, Bevezetés a korpusznyelvészetbe, Budapest: Tinta Könyvkiadó, 2005. |
| [7] | Stanford Core NLP Függőségi Tankönyv, 2015. |
| [8] | B. Liu, Sentiment Analysis and Opinion Mining, Morgan & Claypool Publishers, 2012. |
| [9] | „Stanford Sentiment Analysis honlap,” [Online]. Available: http://nlp.stanford.edu/sentiment/index.html. |
| [10] | R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng és C. Potts, „Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank,” *Stanford University.* |
| [11] | A. Agarwal, B. Xie, I. Vovsha, O. Rambow és R. Passonneau, „Sentiment Analysis of Twitter Data,” *Department of Computer Science, Columbia University,* 2011. |
| [12] | J. Spencer és G. Uchyigit, „Sentimentor: Sentiment Analysis of Twitter Data,” *School of Computing, Engineering and Mathematics, University of Brighton,* 2012. |
| [13] | C. Levallois, „Umigon:Sentiment analysis for tweets based on lexicons and heuristics,” *Department of Marketing Management, Rotterdam School of Management and Erasmus Studio, Erasmus University Rotterdam,* 2013. |
| [14] | M. A. Russell, Mining the Social Web, Sebastopol: O'Reilly Media, 2011. |
| [15] | E. Cambria, B. Schuller, Y. Xia és C. Havasi, „New Avenues in Opinion Mining and Sentiment Analysis,” *IEEE Computer Society,* 2013. |
| [16] | A. Pak és P. Paroubek, „Twitter as a Corpus for Sentiment Analysis and Opinion Mining,” *Universite de Paris-Sud, Laboratoire LIMSI-CNRS,* 2010. |
| [17] | B. A. Huberman, D. M. Romero és F. Wu, „Social networks that matter: Twitter under the microscope,” *Social Computing Lab, Cornell University, Ithaca, NY,* 2008. |
| [18] | M.-C. de Marneffe és D. C. Manning, „The Stanford typed dependencies representation,” *Proceeding CrossParser '08 Coling,* 2008. |
| [19] | S. Wakade, C. Shekar, J. K. Liszka és C.-C. Chan, „Text Mining for Sentiment Analysis of Twitter Data,” *Department of Computer Science, The University of Akron.* |
| [20] | X. Ding és B. Liu, „The Utility of Linguistic Rules in Opinion Mining,” *Department of Computer Science, University of Illinois at Chicago,* 2007. |