
**UNIVERSITATEA SAPIENTIA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE
DEPARTAMENTUL DE MATEMATICĂ - INFORMATICĂ
SPECIALIZAREA DEZVOLTAREA APLICAȚIILOR SOFTWARE**

**Sistem automat de analiză
sentimentală a părerilor despre
vehicule**

Lucrare de disertație

**CONDUCĂTOR ȘTIINȚIFIC
Dr.ing. JOHANN STAN,**

**ABSOLVENT
ASZALOS GYÖRGY**

2015

UNIVERSITATEA SAPIENTIA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE DIN TÂRGU-MUREȘ
SECȚIA DEZVOLTAREA APLICAȚIILOR SOFTWARE

VIZAT DECAN
Dr.Kelemen András

VIZAT ȘEF CATEDRĂ

LUCRARE DE DISERTAȚIE

Conducătorul temei:
Dr.JOHANN STAN

Candidat: **ASZALOS GYÖRGY**
Anul absolvirii : 2015

1. Conținutul proiectului

a) Tema proiectului de diplomă: *Sistem automat de analiză sentimentală a părerilor despre vehicule*

b) Problemele principale tratate:

- Sistemul propus in această lucrare permite rezolvarea problemei de extragere a sentimentului dintr-un mesaj nestructurat, foarte comun pe rețelele sociale.

c) Desene obligatorii:

- Schema bloc al aplicației
- Diagramele de proiectare ale aplicației:activity, class și component
- Structura bazei de date utilizate

d) Softuri obligatorii:

- Aplicație web pentru afișarea rezultatelor în urma analizei sentimentale din tweet-uri.
- Aplicație de tip windows service pentru colectarea tweet-urilor.
- Aplicație de tip windows service pentru analiza tweet-urilor.

Bibliografie recomandată:

1. B. Pang és L. Lee, „Opinion Mining and Sentiment Analysis,” *Foundations and Trends in Information Retrieval*, 2008.
2. M.-C. de Marneffe és D. C. Manning, „The Stanford typed dependencies representation,” *Proceeding CrossParser '08 Coling*, 2008.
3. X. Ding és B. Liu, „The Utility of Linguistic Rules in Opinion Mining,” *Department of Computer Science, University of Illinois at Chicago*, 2007.

Termene obligatorii de consultații: - săptămânal

Locul practicii: Universitatea Sapientia

Primit la data de:

Termen de predare:

Semnătura șefului de catedră

Semnătura îndrumătorului științific

Semnătura candidatului

Declarație

Subsemnata/Subsemnatul, funcția....., titlul științific declar pe propria răspundere că absolventul specializării dea întocmit prezenta lucrare cu îndrumarea mea.

Forma finală a lucrării a fost verificată de mine și aceasta corespunde cu cerințele de formă și conținut precizate de Consiliul Facultății în baza reglementărilor Universității "Sapientia". Lucrarea/proiectul corespunde și cerințelor impuse de Legea Educației Naționale 1/2011 cu modificări ulterioare, Codului de etică și deontologie profesională a Universității Sapientia referitoare la furt intelectual.

Sunt de acord cu susținerea lucrării în fața comisiei de examen de disertație.

Localitatea,

Data:

Semnătura îndrumătorului,

Declarație

Subsemnata/ul, absolvent(ă) al/a specializării, promoția, cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapiientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data:

Absolvent

Semnătura.....

Extras

Cuvinte cheie: big data, extragerea de cunoștințe din date, media de socializare, prelucrarea limbajului natural, analiza sentimentelor

În această lucrare este prezentat un sistem care își construiește propria bază de date din tweet-uri aduse de pe rețeaua de socializare Twitter, prelucrează și efectuează analiza sentimentelor asupra textului acestor tweet-uri. Tweet-ul este un mesaj textual cu o lungime de maxim 140 de caractere, care este vizibil pentru oricare alt utilizator de Twitter.

Tema și scopul lucrării

În zilele noastre datele au un volum uriaș, îmbracă diverse forme, circulă cu viteze foarte mari și nu întotdeauna sunt adevărate și din ce în ce mai mulți utilizatori folosesc mediile sociale, din toate părțile lumii. Pe baza acestor fapte ar fi interesant și ar merita să aflăm ce gândesc, care este opinia utilizatorilor despre diverse lucruri, produse sau chiar persoane. Acest lucru ar putea fi posibil dacă s-ar analiza opiniile despre anumite proprietăți ale produsului. Ar fi o informație utilă, atât pentru consumatori, cât și pentru producătorii produsului, ce proprietăți ale produsului le plac și ce proprietăți nu le plac utilizatorilor.

Studiu bibliografic

În prelucrarea datelor, extragerea de cunoștințe din date are un rol important. În legătură cu extragerea de cunoștințe din date s-au formulat următoarele idei: transformarea unui volum de date în cunoștințe, și descoperirea de cunoștințe din date reprezentând o etapă în procesul de descoperire a cunoștințelor.

Prin prelucrarea limbajului natural se prelucrează textul, iar cuvintele vor primi anotații de parte de vorbire și sintactice.

Analiza sentimentelor este acel domeniu al științei, care analizează părerile, sentimentele, atitudinile oamenilor. Aceste opinii pot fi formulate despre produse, persoane, firme. Majoritatea sistemelor de analiză a sentimentelor funcționează având la bază un lexicon al sentimentelor predefinit, pe baza căruia se decide, despre o anumită opinie dacă este pozitivă sau negativă. Lexiconul sentimentelor conține cuvinte, care sunt în relație cu un anumit sentiment.

Fundamentarea teoretică

Reprezentarea dependențelor Stanford oferă o descriere simplă a relațiilor gramaticale dintr-o propoziție. Definițiile de dependență ale procesatorului de texte Stanford folosesc anotațiile Penn – Treebank part-of-speech (parte de vorbire) și sintactice. Dependențele pot fi vizualizate în 5 stiluri: de bază, restrâns, restrâns cu propagare, restrâns cu păstrarea structurii de arbore, ne-restrâns. În urma anotării părților de vorbire în propoziție, lângă fiecare cuvânt va apărea anotația

despre partea de vorbire. Pentru căutarea cu succes a cuvintelor cheie am introdus posibilitatea definirii de sinonime pentru aceste cuvinte cheie, deoarece un anumit vehicul poate fi exprimat prin mai multe cuvinte. În cazul proprietăților am luat în considerare forma comună canonică (lematizare) a cuvintelor.

Pe parcursul căutării relațiilor cuvânt cheie – proprietate, bazându-mă pe observațiile mele, am presupus că în propoziție atât proprietatea cât și cuvântul cheie vor fi anotate ca substantiv. Folosindu-mă de această presupunere, am stabilit distanțele dintre cuvintele cheie din graful de dependență. În graf am considerat că fiecare muchie are lungimea de 1. Urmând ca, pentru fiecare cuvânt cheie cunoscut algoritmul meu să stabilească lungimea drumului către proprietatea cea mai apropiată. Dacă de proprietate nu se află alt cuvânt cheie mai apropiat, decât cuvântul cheie actual, atunci s-a găsit cu succes relația cuvânt cheie – proprietate. Dacă spre proprietate și din altă proprietate există drum de aceeași lungime cu lungimea către cuvântul cheie, atunci cuvântul cheie se află în relație și cu această proprietate.

Pentru stabilirea sentimentelor am folosit motorul de stabilire a sentimentelor oferit de Stanford Core NLP . Această componentă folosește o rețea neuronală recursivă tensor pentru stabilirea sentimentelor și construiește structura arborelui sentimentelor pentru propoziție. Pentru fiecare nod din acest arbore al sentimentelor se va atribui o notă ce cuprinde cinci valori (foarte negativ, negativ, neutru, pozitiv, foarte pozitiv) procentuale.

Pentru a putea stabili valoarea sentimentelor aferentă relației găsite, am definit următoarea regulă: voi alege nodurile din arborele sentimentelor care conțin cuvântul cheie, proprietatea și un verb. Algoritmul de stabilire a sentimentelor în cazul fiecărei relații va parcurge arborele sentimentelor în mod post-order. Valoarea sentimentelor primului nod care îndeplinește condiția anterior prezentată, va fi atribuită relației actuale.

Arhitectura sistemului, proiectarea și implementarea

Sistemul elaborat de mine cuprinde o aplicație web, două componente de servicii, o componentă Data access și o componentă model. Prin aplicația web, utilizatorul are acces la modulele de administrare și vizualizare a rezultatelor. Cele două servicii efectuează descărcarea tweet-urilor, respectiv analiza sentimentelor. Cu ajutorul claselor din componenta Data access citim și modificăm datele stocate în baza de date. Componenta model conține clasele corespondente tabelor din baza de date. Sistemul este alcătuit din următoarele module: modulul de colectare a tweet-urilor, modulul de analiză a textului, modulul de analiză a sentimentelor, modulul de administrare și modulul de vizualizare a rezultatelor.

Rolul modulului de colectare a tweet-urilor este acela de a descărca tweet-urile care conțin cuvintele cheie definite de utilizator și salvarea lor în baza de date. Din cauza limitărilor impuse

de Twitter API, se programează desfășurarea în timp a descărcărilor, asigurând continuitatea procesului, fără a fi nevoie de intervenția utilizatorului.

Modulul de analiză al textului efectuează anotația părți-de-vorbire a cuvintelor și analizează relațiile gramaticale dintre cuvintele propoziției.

Modulul de analiză a sentimentelor efectuează evaluarea sentimentelor din tweet-uri și salvează rezultatele obținute în baza de date. Modulul de analiză a sentimentelor funcționează pe baza algoritmului prezentat anterior: prima dată identifică relațiile cuvânt cheie – proprietate, iar pe urmă evaluează sentimentele folosind Stanford Parser. Pe baza acestora i se va stabili o notă proprie ce reflectă valoarea sentimentului aferent perechii cuvânt cheie – proprietate.

Modulul de administrare este o aplicație web, cu ajutorul căruia utilizatorul poate defini listele cu cuvintele-cheie, sinonime și proprietăți. Poate porni, respectiv opri serviciile de colectare a tweet-urilor și analiză a sentimentelor.

Modulul de vizualizare a rezultatelor face parte tot din aplicația web. Prin acest modul, utilizatorul poate alege categoria, cuvintele-cheie și proprietățile, despre care vrea să vizualizeze rezultatele obținute în urma evaluării sentimentelor. Rezultatele vor fi prezentate într-un tabel comparativ. Pentru o pereche cuvânt-cheie – proprietate se pot vizualiza textele a cinci tweet-uri și rezultatele evaluării sentimentelor efectuate de Stanford Parser.

Testarea sistemului și rezultatele obținute

Deoarece identificarea relației dintre cuvântul cheie - proprietate reprezintă una dintre cele mai importante funcții ale sistemului, eficiența acestuia a fost testată cu ajutorul unor unit test-uri. Cazurile testate au fost alcătuite din propoziții în care am variat numărul cuvintelor cheie cunoscute, proprietățile cunoscute, cuvintele cheie reale și proprietățile reale. Cazurile testate au fost considerate ca fiind cu succes dacă s-au identificat toate relațiile reale și nu s-au găsit relații false. Din cele 45 de cazuri testate, algoritmul de identificare a relațiilor a găsit corect relațiile cuvânt cheie – proprietate din propoziție, în 39 de cazuri. Asta înseamnă 86.66% rată de identificare corectă.

Concluzii

Utilitatea sistemului constă în faptul că oferă posibilitatea colectării tweet-urilor ce conțin cuvintele cheie predefinite. În urma analizei sentimentelor se obțin informații utile cu privire la ce gândesc utilizatorii despre o anumită proprietate. În cazul mărcilor de vehicule putem vedea ce proprietate ce rezultat a obținut, în ce este bun, în ce este mai slab. Această informație este utilă pentru utilizatori deoarece află ce mărci au obținut rezultate mai bune și pentru ce proprietăți. Asta îi poate ajuta în decizia de cumpărare a unui vehicul. Pentru producătorii de vehicule de asemenea este utilă această informație, deoarece pe baza lor pot lua decizii despre ce anume

trebuie îmbunătățit, identifica de ce sunt nemulțumiți utilizatorii și concurența unde a obținut rezultate mai bune.

Sistemul poate fi dezvoltat în continuare în următoarele părți: utilizarea Stream API pentru colectarea tweet-urilor, reevaluarea tweet-urilor colectate și evaluate deja, lărgirea bazei de colectare a datelor cu site-uri de review, pagini web și alte rețele de socializare, precum și îmbunătățirea aplicației web cu un modul de vizualizare a stării sistemului.

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MŰSZAKI ÉS HUMÁNTUDOMÁNYOK KAR
MATEMATIKA-INFORMATIKA TANSZÉK
SZOFTVERFEJLESZTÉS SZAK**

Járművekről szóló vélemények automatikus érzelemelemző rendszere

Mesteri disszertáció

**TÉMAVEZETŐ,
Dr. JOHANN STAN**

**VÉGZŐS HALLGATÓ,
ASZALOS GYÖRGY**

2015

Kivonat

Kulcsszavak: big data, adatbányászat, közösségi média, szövegelemzés, érzelelemzés

Ez a dolgozat egy olyan rendszert mutat be, amely a Twitter közösségi háló csevegéseiből épít fel egy saját adatbázist és ezeknek a tweeteknek a szövegén végez szöveg- és érzelelemzést. A tweetek rövid szöveges üzenetek, maximum 140 karakter hosszal, melyek láthatóak bármely más Twitter felhasználó számára

A dolgozat témája és céljai

Napjainkban az adatok óriási tömegűek, változatosak, nagy sebességgel terjednek és nem mindig igazmondóak. Egyre többen használják a közösségi médiákat, a világ minden tájáról. Az előbb említett tények alapján érdekes és érdemes lenne megtudni mit is gondolnak, mi a véleményük a felhasználóknak különböző dolgokról, termékekről vagy akár személyekről. Ezt azzal fokozhatjuk, ha az adott termék bizonyos tulajdonságairól megfogalmazódott véleményeket elemezzük. A termék felhasználóinak és gyártóinak egyaránt hasznos információt jelentene, hogy a felhasználók a termék mely tulajdonságait kedvelik és melyeket nem.

Bibliográfiai tanulmány

Az adatok feldolgozásában fontos szerepe van az adatbányászatnak. Az adatbányászat fogalmával a következő nézőpontok fogalmazódtak meg: átalakít egy adattömeget tudássá, ismeretek felfedezése adatokból és egy lépés az ismeret felfedezésének a folyamatában.

A számítógépes szövegelemzés során feldolgozódik a szöveg, majd a szavak szófaji, szintaktikai vagy mondattani címkékkel lesznek megjelölve.

Az érzelelemzés az a tudományterület, mely az emberek véleményét, érzelmét, attitűdjét elemzi. Ezek a vélemények megfogalmazódhatnak termékekről, személyekről, cégekről. Az érzelelemző rendszerek nagy része úgy működik, hogy egy előre definiált érzelemlexikont használnak, mely alapján eldöntik, hogy egy adott vélemény pozitív vagy negatív. Az érzelemlexikon olyan szavakból áll, melyekhez valamilyen érzelem kapcsolódik.

Elméleti megalapozás

A Stanford függőségi reprezentáció egy egyszerű leírást ad a mondatban szereplő nyelvtani kapcsolatokról. A Stanford szövegelemző függőségi definíciói a Penn – Treebank part-of-speech (szófaji) és mondattani címkéket használják fel. A függőségek ábrázolása a Stanford Parser használata esetében 5 stílusban lehetséges: alap, összevont, propagálódó összevont, összevont fa szerkezetet megőrző, nem-összevont. A szófaji címkézés során a mondatban minden egyes szó mellett megjelenik a szófaji megjelölés. A névelemek sikeres keresése érdekében bevezettem a

kulcsszavak lebontását szinonimákra, hisz ugyanazt a járművet különböző szavak is jelölhetik. A tulajdonságok esetében szavaknak a közös kanonikus (lemmatizált) alakját vettem figyelembe.

A tulajdonság – névelem relációk megkeresése során, a megfigyeléseimre alapozva, feltételeztem, hogy a mondatban a tulajdonság és a névelem is főnévi címkével lesz jelölve. Ezt a feltételezést felhasználva, meghatároztam a távolságot a függőségi gráfban szereplő névelemek között. A gráfban az élek hosszát 1-nek tekintettem. Ezt követően minden ismert névelemre az algoritmusom 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 tulajdonságból is vezet ugyanolyan távolságú út, mint az éppen aktuális névelemhez lévő távolság, akkor a névelem azokkal a tulajdonságokkal is relációban áll.

Az érzelem meghatározására a Stanford Core NLP motor érzelem-meghatározó komponensét használtam. Ez a komponens egy rekurzív tenzor neuronhálót használ az érzelmek meghatározására és strukturálisan felépíti a mondatnak megfelelő érzelmi fát. Az érzelmi fa minden csomópontjának megfeleltet egy ötös értéket (nagyon negatív, negatív, semleges, pozitív és nagyon pozitív), százalékosan kifejezve.

Ahhoz, hogy a meg talált relációknak az érzelmi értékét meg tudjam határozni, a következő szabályt definiáltam: azokat a csomópontokat választom 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 az érzelmi fa csomópontjain. Az első olyan csomópontnak az érzelmi értékét feleltem meg az aktuális relációnak, mely teljesítette az előbbi szabályt.

A rendszer architektúrája, tervezése és megvalósítása

Az általam elkészített rendszer tartalmaz egy web applikációt, két szerviz komponenst, egy Data access komponenst és egy modell komponenst. A web applikáción keresztül éri el a felhasználó az adminisztrációs és az eredményjelző modult. A két szerviz végzi a tweetek letöltését, valamint az érzelemelemzést. A Data access komponensben lévő osztályok segítségével olvassuk ki és módosítjuk az adatbázisban tárolt adatokat. A modell komponens tartalmazza az adatbázisban szereplő tábláknak osztályokra való leképezését. A rendszer moduljai a következők: tweet-gyűjtő modul, szövegelemző modul, érzelemelemző modul, adminisztrációs modul és eredményjelző modul.

A tweet gyűjtő modul szerepe a felhasználó által definiált kulcsszavakat tartalmazó tweetek letöltése és elmentése a saját adatbázisba. A Twitter API tweet letöltési korlátai miatt, a tweet

letöltéseknek az időzítését ütemezi, így a letöltés folyamatos lesz, nem igényelve felhasználói beavatkozást.

A szövegelemző modul a tweetek szövegén végez szófaji annotációt és a mondatban a szavak közti nyelvtani függősségi kapcsolatokat elemzi ki.

Az érzelelemlemző modul a letöltött tweeteken végez érzelmi kiértékelést és a kiértékelt eredményeket elmenti az adatbázisba. Az érzelelemlemző modul az előbb bemutatott algoritmus szerint működik: először megkeresi a névelem-tulajdonság relációkat, majd elvégzi ezeken az érzelelemlemzést a Stanford Parser segítségével. Ezek alapján egy saját jeggyel látom el az adott névelem – tulajdonság páros érzelmi értékét.

Az adminisztrációs felület egy web applikáció, amellyel a felhasználó definiálhatja a kulcsszavak, szinonimák, tulajdonságok listáját. Elindíthatja, valamint megállíthatja a tweetgyűjtő és érzelelemlemző szervizeket.

Az eredményjelző modul ugyancsak a web applikáció része. Ezen a felületen a felhasználó kiválaszthatja azt a kategóriát, kulcsszavakat és tulajdonságokat, amelyekre a kiértékelt eredményeket szeretné megtekinteni. Az elért eredményeket egy összehasonlító táblázat mutatja be. Egy adott kulcsszó – tulajdonság párosra megtekinthető öt tweet szövege és a Stanford Parser érzelelemlemző által kiértékelt eredmények.

A rendszer tesztelése és eredmények

Mivel a rendszer egyik legfontosabb feladata a névelem – tulajdonság közti reláció meghatározása, ezért ennek a hatékonyságát unit tesztek segítségével ellenőriztem. A felállított tesztesetek olyan mondatokból álltak melyekben, az ismert kulcsszavak, ismert tulajdonságok, valódi kulcsszavak és valódi tulajdonságok különböző számban való megjelenése volt variálva. A tesztesetek sikeressége azt jelentette, ha megtaláltuk az összes valódi relációt és nem találtunk hamis relációkat. A 45 teszt esetében a relációkereső komponens 39 esetben helyesen állapította meg a mondatban szereplő kulcsszó-tulajdonság relációt. Ez 86.66%-os helyes meghatározást jelent.

Következtetések

A rendszer hasznossága abban nyilvánul meg, hogy lehetőséget ad a meghatározott kulcsszavakat tartalmazó tweetek összegyűjtésére. Az érzelelemlemzés után hasznos információkat kapunk arról, hogy a felhasználók mit is gondolnak, az adott tulajdonságról. Az autómárkák esetében láthatjuk, hogy milyen tulajdonságra milyen eredményt ért el, miben jó, miben gyengébb. Ez az információ hasznos a vásárlók szempontjából hisz megtudhatják mely autómárkák értek el jobb eredményt és melyik tulajdonságokra. Ez segíthet dönteni autóvásárlás esetén. Az autógyárak szempontjából is hasznos ez az információ, hisz ezek alapján hozhatnak

döntést arról, hogy mit kell feljavítani, mivel elégedetlenek a felhasználók, miben ért el a konkurencia jobb eredményt.

A rendszert a következő részein lehet továbbfejleszteni: a tweetek gyűjtését a Stream API használatával, a letöltött és kiértékelt tweetek újra – kiértékelését, az adatforrások bővítését review-s, web – oldalak és más szociális hálókkal valamint a web applikáció bővítését egy rendszerállapot kijelző modullal.

**SAPIENTIA HUNGARIAN UNIVERSITY OF TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
DEPARTMENT OF MATHEMATICS-INFORMATICS
SPECIALIZATION SOFTWARE DEVELOPMENT**

Automated sentiment analysis system of opinions about vehicles

Master's Thesis

**SUPERVISOR,
Dr. JOHANN STAN**

**STUDENT,
ASZALOS GYÖRGY**

2015

Abstract

Keywords: big data, data mining, social media, text analysis, sentiment analysis

This work presents a system, which builds an internal database of tweets obtained from the Twitter social network and performs text and sentiment analysis on these. By definition, a tweet is a small text message, with a maximum length of 140 characters, visible for any Twitter user.

The topic and purpose of the dissertation

Nowadays the volume of the data is big, can have many representations, is streamed with high speed and is sometimes unreliable. The number of social media users is continuously growing, with data being shared from every part of the world. Because of this fact it would be interesting and useful to know what the users think, what is their opinion about things, products or even other people. This can be improved if we analyze the opinions about some properties of a given entity, such as a product. Both for product customers and manufacturers it would be useful to know which properties of the product are appreciated and which are not.

Bibliographic study

In the process of text analysis data mining has an important role. There are various opinions about the definition for data like: transform a given volume of data into knowledge, knowledge discovery from data, a step in the knowledge discovery process.

After performing text analysis using a natural language parser, we will get as result a text with part-of-speech and syntactical annotation.

Sentiment analysis is a field of science which analyzes the opinions, sentiments or attitudes of people. These can be opinions about products, people or companies. The majority of sentiment analyzers use a predefined sentiment lexicon, which is then used to evaluate if an opinion is positive or negative. The sentiment lexicon contains words, to which some kind of sentiment is associated.

Theoretical foundations

The Stanford typed dependency representation is a simple description of the grammatical relationships in a sentence. The typed dependency definitions of the Stanford parser use the Penn – Treebank part-of-speech and syntactical annotations. The typed dependency representation of the Stanford Parser can be: basic, collapsed, collapsed with propagation, collapsed with tree structure and non-collapsed. After performing part-of-speech annotation every word of the sentence will have a part-of-speech annotation attached. In order to improve the keyword search, I defined synonyms for keywords, because a given vehicle can be defined with different words. In the case of properties I used the common canonical (lemmatized) form of words.

Through the process of finding relations between a keyword and property, based on my observation, I assumed that both the keyword and property has a noun annotation. Using this assumption, I calculated the distance between keywords in the typed dependency graph. In the graph I consider the length of an edge to be 1. In the next step for every known keyword my algorithm calculates the distance to the nearest known property. If there is no other keyword closer to the property, than the actual keyword, it means the algorithm has successfully found the keyword-property relation. If there exists a path with the same length, to another property, from the property related to the keyword, it means the other property is in relation with the keyword.

For the sentiment analysis I used the Stanford Core NLP sentiment tool. This component uses a recursive neural tensor network and builds sentiment treebank for sentences. Every node in the sentiment treebank has five values (very negative, negative, neutral, positive, very positive), expressed by percentage.

In order to define the sentiment value of a relation I defined the following rule: pick that node from the sentiment treebank, which contains the keyword, the property and a verb. The sentiment analyzer algorithm traverses the treebank nodes in post-order mode. The sentiment value of the first node, that satisfies the above defined rule, will serve as result for the actual relation.

System architecture, planning and implementation

The implemented system contains a web application, two service components, a Data access component and a model component. Through the web application the user has access to the administration and result visualization modules. The two service components collect the tweets and perform the sentiment analysis. The classes from the Data access component are used to read and modify the data from the database. The model component contains the object-relational mapping of the database tables into classes. The system can be divided into the following modules: tweet-collector module, text analyzer module, sentiment analyzer module, administration module and result visualization module.

The role of the tweet collector module is to search, download and save the tweets which contain the predefined keywords into the database. Because of a Twitter API limitation, the tweet downloads are scheduled, in order to have a continuous download, so there is no need for user intervention.

The text analyzer module makes the part-of-speech annotation over the words and finds the typed dependencies.

The sentiment analyzer module performs the sentiment evaluation on the collected tweets and saves the evaluated results into the database. The sentiment analyzer module uses the algorithm I have described earlier: it first searches for the keyword-property relations, followed by the

sentiment evaluation using the Stanford Parser. Based on these evaluations I give a mark for the keyword-property relation sentiment result.

The administration module is a web application, where the user can define the keywords, synonyms, and properties. The user has the possibility to start or stop the tweet collector and sentiment analysis services.

The result visualization module is also part of the web application. Here the user can choose a category, keywords and properties, for which the evaluated results will be displayed. The evaluated results will be shown in form of a comparison table. For a given keyword-property pair, the system can display five related tweets and the evaluated results by the Stanford Parser.

System testing and results

Because the task to find the keyword-property relations is a key function of the system, I have written unit tests to test its efficiency. The test cases consist of sentences which contain various words like: known keywords, known properties, real keywords and real properties. I consider a test case to be successful if all real relations are found and no false relations.

From the 45 test cases the relation finder component has found the right keyword-property relations from a sentence in 39 cases. This is 86.66% rate of relation finding success.

Conclusions

This system is helpful, because it offers the possibility to collect tweets containing predefined keywords. After the sentiment evaluation we obtain useful information about what the users think about a given property. In the case of cars and their manufacturers, we can get an overview about which car excels or fails at which characteristic. This information is useful for users, because they can find out which cars have better results and for what characteristics, and based on these results they can decide which car to buy. For the car manufacturer this information can serve as basis for decision making about what needs further improvement, why the users are unsatisfied and which are the domains where the competition obtained better results.

Tartalomjegyzék

1	Bevezető.....	1
2	Bibliográfiai tanulmány	3
2.1	Adatgyűjtés	3
2.2	Szövegelemzés	3
2.3	Érzelelemlemzés	5
3	Elméleti megalapozás	6
3.1	Stanford Parser, Mondattani elemzés	7
3.2	Névelemek (kulcsszavak) keresése	9
3.3	Tulajdonságok keresése	9
3.4	Tulajdonság – névelem relációk felismerése	10
3.5	Érzelem meghatározás.....	15
4	A rendszer megvalósítása.....	17
4.1	A rendszer specifikációi és architektúrája	17
4.2	A részletes tervezés	18
4.2.1	UI komponens	18
4.2.2	Adat-gyűjtő komponens	23
4.2.3	Elemző komponens	35
4.2.4	Data access komponens	45
4.2.5	A model komponens.....	47
4.3	A rendszer felhasználása	47
4.4	Üzembe helyezés és kísérleti eredmények	48
4.4.1	Felhasznált technológiák	48
4.4.2	Felmerült problémák és megoldásai.....	49
4.4.3	A relációkeresés eredményei	50
5	Következtetések.....	51
5.1	Megvalósítások	52

5.2	Hasonló rendszerekkel való összehasonlítás	52
5.3	További fejlesztési irányok.....	54
5.3.1	Tweetek gyűjtése Stream API-val.....	54
5.3.2	Tweet szöveg újra - kiértékelése.....	54
5.3.3	Adatforrások bővítése.....	55
5.3.4	Rendszerállapot kijelzés	55
6	Irodalomjegyzék	56

Ábrák jegyzéke

1. Ábra Adatbányászat folyamata	4
2. Ábra Függőségek diagram	11
3. Ábra Alap (a) és összevont (b) függőségek	12
4. Ábra Példamondat összevont függőségi gráfja	14
5. Ábra Érzelmi fa	16
6. Ábra Rendszer funkcionális architektúra.....	18
7. Ábra Rendszer architektúra.....	18
8. Ábra Szervizek üzemeltetése	19
9. Ábra Kategóriák lista.....	19
10. Ábra Új kategória létrehozása	20
11. Ábra Kategória módosítása	20
12. Ábra Kategória törlése	21
13. Ábra Érzelelemzési eredmények szűrése	21
14. Ábra Érzelelemzési eredmények	22
15. Ábra Érzelemeredményhez tartozó tweetek.....	22
16. Ábra Egyed-kapcsolat diagram	28
17. Ábra TweetDownloader osztály diagram kapcsolatok	28
18. Ábra TweetDownloaderService osztály diagram.....	29
19. Ábra TweetDownloaderService activity diagram	30
20. Ábra TweetFinderCoordinator osztály diagram.....	31
21. Ábra TweetFinderCoordinator activity diagram	32
22. Ábra TweetFinder osztály diagram	33
23. Ábra TweetFinder activity diagram.....	34
24. Ábra ILogger és Logger osztály diagram	35
25. Ábra TweetAnalyzer osztály diagram kapcsolatok	36
26. Ábra TweetAnalyzerService osztály diagram.....	36
27. Ábra TweetAnalyzerService activity diagram	37
28. Ábra TweetAnalyzerCoordinator osztály diagram.....	38
29. Ábra TweetAnalyzerCoordinator activity diagram	39
30. Ábra TweetAnalyzer osztály diagram	41
31. Ábra TextAnalyzer osztály diagram.....	41
32. Ábra TextRelationFinder osztály diagram.....	43

33. Ábra TextSentimentFinder osztály diagram	44
34. Ábra Segéd modell osztályok.....	44
35. Ábra CategoryDAO osztály diagram.....	45
36. Ábra CharacteristicDAO osztály diagram	45
37. Ábra KeywordDAO osztály diagram	46
38. Ábra SynonymDAO osztály diagram.....	46
39. Ábra TweetDAO osztály diagram	46
40. Ábra EvaluatedResultDAO osztály diagram	47
41. Ábra WinServicesDAO osztály diagram.....	47
42. Ábra Teszteredmények sikeressége.....	51

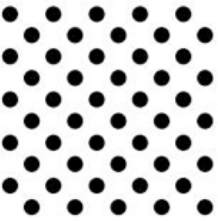
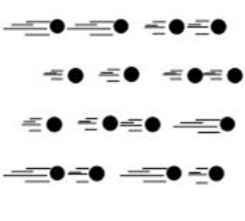
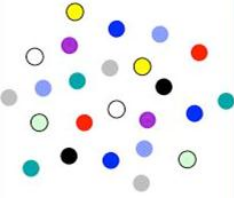
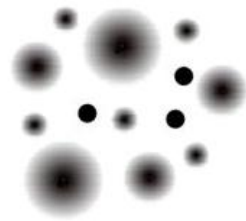
Táblázatok jegyzéke

1. Táblázat A „Big data” négy tulajdonsága a 4V :	1
2. Táblázat Stanford Parser függőségi ábrázolások	8
3. Táblázat Stanford Parser által használt szófaj-annotációk	8
4. Táblázat Lemmatizálás és szótövezés	10
5. Táblázat Függőségi példák.....	11
6. Táblázat Pszeudokód magyarázata.....	13
7. Táblázat Távolság mátrix.....	14
8. Táblázat Érzelem értékek.....	16
9. Táblázat Tweet felépítése	23
10. Táblázat Adatbázisban szereplő táblák listája.....	27
11. Táblázat Relációkeresés teszt eredmények	50

1 Bevezető

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

1. Táblázat 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, milimásodpercektől másodpercekig tartó válaszidő	Strukturált, strukturálatlan, szöveg, multimédia	Bizonytalanság ami az adat inkonzisztenciájából 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 üzletükről 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 asszociálják a következő két ötlettel: adat tárolás és adat elemzés. [2]

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

Érzelelemelemzés

Mindig kíváncsiak 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 megfogalmaznak érzésekből fakadnak. Az érzések szubjektív benyomások nem pedig tények. Az érzések hozzáállásban, érzelmekben és véleményekben nyilvánulhatnak ki. Az érzelelemelemzés felhasználva szövegelemzőket, statisztikákat vagy géptanulási technikákat ki bányász, meghatároz, vagy másképp fogalmazva jellemzi az érzelmi tartalmát egy adott szövegnek. Bizonyos esetekben úgy is hivatkoznak az érzelelemelemzésre, mint véleménykutatás. Az érzelelemelemzés a következő kérdéseket fogalmazhatja meg:

- Pozitív vagy negatív a termékről szóló véleményezés?
- Ez a vásárló mailje elégedettséget vagy elégedetlenséget tükröz?
- Hogy változott a bloggerek hozzáállása az elnökhöz a választások óta?

Az érzelelemelemzés alkalmazható az üzleti világban. Választ adhat olyan kérdésekre, mint: Miért a vásárlók, nem a mi termékünket veszik meg? Meg szeretnénk tudni a szubjektív 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 hálókön vagy a közösségi médiákban keresni véleményeket a saját termékünkről vagy akár a konkurenseink termékéről. [5]

Az általam készített applikáció a Twitter – ismeretségi hálózat csevegéseiből épít fel egy saját adatforrást. A csevegések szövegét mondattani elemzéssel feldolgozza, megkeresi a kulcsszó – tulajdonság kapcsolatokat és az így felépített mondatrészre érzelmi kiértékelést végez. Az

applikáció segítségével a felhasználó értékes információt kap az ügyfelek érzelmeiről az adott termék bizonyos tulajdonságairól, ily 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 konkurens 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.

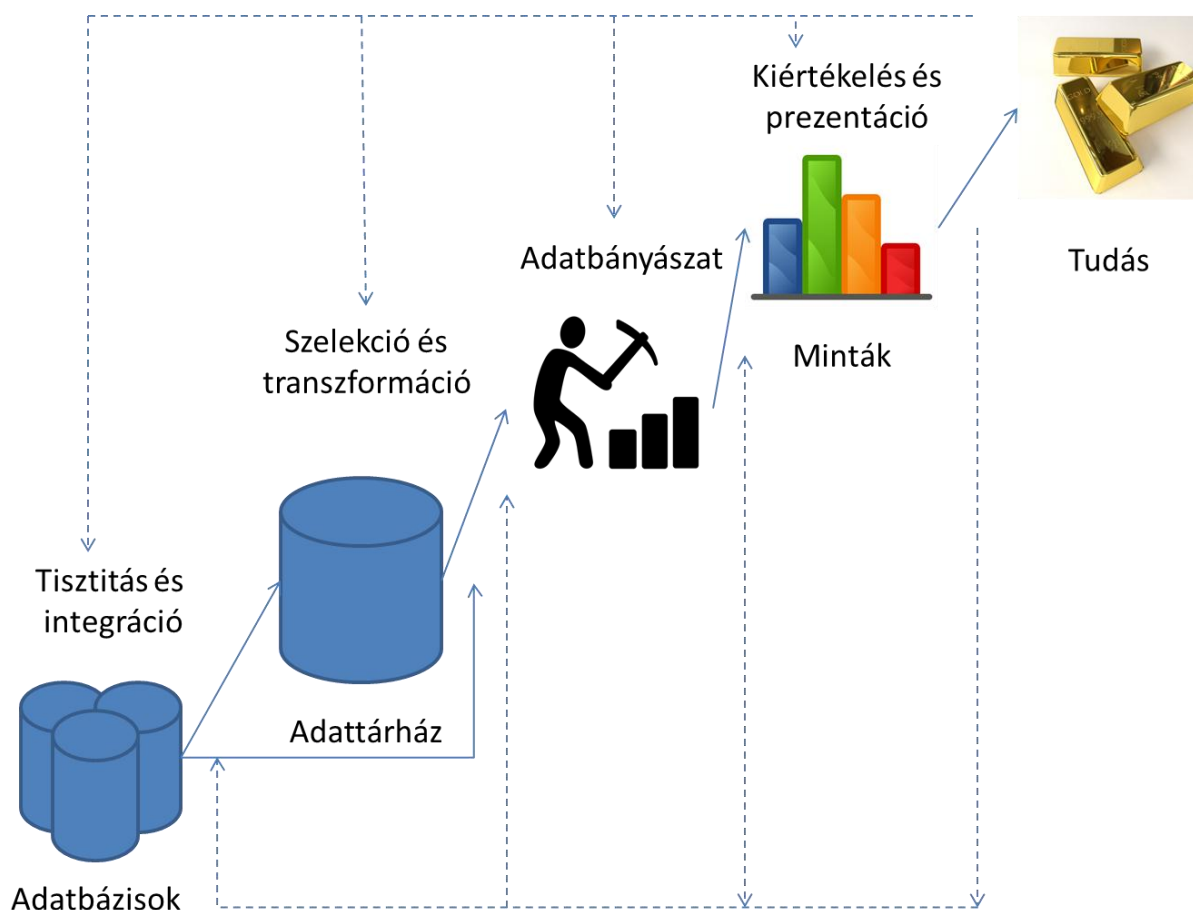
2 Bibliográfiai tanulmány

2.1 Adatgyűjtés

„Az információ korában élünk” ezt tartja a népszerű mondás napjainkban, pedig inkább azt mondhatnánk, hogy az adat korában élünk. Terra bájtt vagy peta bájt nagyságú adatok özönlének 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. [6] A közösségek és közösségi média adatának a fontossága egyre növekedik és változatos formájú adatokat hoz létre, mint: digitális képek és videók, blog-ok, Web közösségek, és különböző típusú szociális hálózatokat. Erőteljes, sokoldalú és flexibilis eszközökre van szükség, amelyek automatizáltan kiemeljenek értékes információkat ezekből a hatalmas adattömegekből és átalakítsák 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. 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 állandóan folyamló adatokat egy dinamikus rendszerbe.

2.2 Szövegelemzés

A természetes nyelv feldolgozásának fő célja természetes nyelvet megértő és feldolgozó applikációk készítése. Természetes nyelv alatt emberi lények által beszélt vagy írott nyelvet értünk, ellentétben a mesterséges, számítógépi, számtani vagy logikai nyelvvel, például. [7]



1. Ábra Adatbányászat folyamata

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. [8]

Itt említeném meg a korpusz annotációt, 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. [9] 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 mellett 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 kategóriájába.

Egy természetes nyelv szövegelemző egy olyan applikáció, amely előállítja a mondatok nyelvtani struktúráját, mely szó-csoportok alkotnak együtt egy frázist, és mely szavak az alany vagy tárgy részei egy igének. [10]

2.3 Érzelelemzés

Az érzelem analízis, más néven véleménybányászat, az a tudományterület mely az emberek termékekről, szolgáltatásokról, szervezetekről, egyénekről, eseményekről, témakörökről alkotott véleményét, érzelmét és attitűdjét elemzi [11]. Ez a tudományterület több néven 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. [12]

Az érzelelemelemzést szintekre lehet bontani. Az első szint a dokumentum szintű érzelelemelemzés. Ennek során egy egész dokumentum alapján dönti el a rendszer, hogy összességében a vélemény pozitív vagy negatív. A második szint a mondat szintű érzelelemelemzés. 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 autót”. 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.”

3 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.

3.1 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 hatékonyan használható legyen olyan személyek által, akik szövegek közti kapcsolatokat akarnak kinyerni anélkül, hogy magas nyelvtani szaktudással rendelkeznének. A jelenlegi Stanford szövegelemző függőségi definíciói megközelítőleg 50 nyelvtani kapcsolatot tartalmaznak. A függőségek mind bináris kapcsolatok: a nyelvtani kapcsolat két részből áll, a vezető (irányító, fej) és az alárendelt (függő). A függőségi definíciók felhasználják a Penn – Treebank part-of-speech és mondattani címkéket.

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

- Elemzési fák [10]

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

- alap
- összevont
- propagálódó összevont
- összevont fa szerkezetet megőrző
- nem-összevont

A különböző függőség típusok bemutatására a következő példamondatot fogom használni:

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

Az alapfüggőségek a Penn Treebank által definiált szófaj és fráziscímkézést használják, és fa szerkezetet alkotnak. A mondatban 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 előjárószók, kötőszók, valamint az információk a kapcsolódó 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 propagálódó összevont függőségek esetében, azokban a mondatokban ahol létezik kötőszó, a függőség propagálódik a kötőszó által összekötött szavakra. A mi példamondatunk esetében a két ige közötti kötőszó miatt, az összevont függőséghez két új függőséget kéne létrehozni. A két ige “makes” és “distribute” között levő kötőszó miatt, az alany és tárgy kapcsolat ami a

kötőszó első elemére, a “makes”-re, vonatkozik tovább kéne propagálódjon a kötőszó második elemére is, a “distribute” –ra:

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 megőrző 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 struktúrát, viszont nem propagál és össze sem von semmiféle kapcsolatokat. [13]

2. Táblázat Stanford Parser függőségi ábrázolások

Alap	Összevont	Propagálódó összevont	Összevont fa	Nem-összevont
nsubj(makes,Bell)	nsubj(makes,Bell)	nsubj(makes,Bell)	nsubj(makes,Bell)	nsubj(makes,Bell)
		nsubj(distributes,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)

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

3. Táblázat Stanford Parser által használt szófaj-annotációk

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 marker	VB	Ige, alap formában
MD	Módosítószó	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		

3.2 Névelemek (kulcsszavak) keresése

A névelemek kereséséhez 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 applikáció a szinonimák bevezetésével oldja meg ezt a szükségletet. Tehát például a Volkswagen autó-márka névelemhez hozzárendelhetünk több szinonimát, mint Golf, Tiguan, Passat, Transporter 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.

3.3 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:

- nyelvészeten lemmatizálás, melynek során értelmes szóalakot állít elő

- alkalmazásorientált számítógépes nyelvészeti szakirodalom (szó)tövezés, melynek során a szó szótövét határozzuk meg, ilyenkor a szó csonkolása történik és nem mindig kapunk értelmes szótári alakot.

A 4. táblázatban néhány példát tekinthetünk meg a két módszerről és a visszatérített eredményekről:

4. Táblázat Lemmatizálás és szótövezés

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

3.4 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

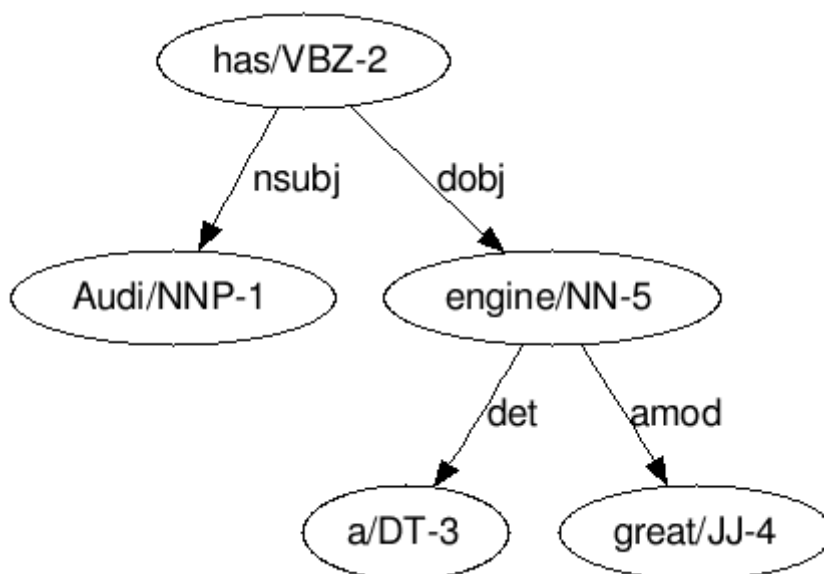
rendszer számára ismert a „BMW” névelem és a „brake” tulajdonság. Ebben az esetben nem létezik reláció az ismert névelem és tulajdonság között, mivel az ismert „brake” tulajdonság az ismeretlen „VW” névelemmel, valamint az ismeretlen „engine” az ismert „BMW” névelemmel áll relációban. Fontos, hogy az 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 megérthetők, akiknek nem szakterületük a nyelvészet [10]. 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:

5. Táblázat Függőségi példák

Függőségek	Stanford Core NLP Tankönyv leírás [10]
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

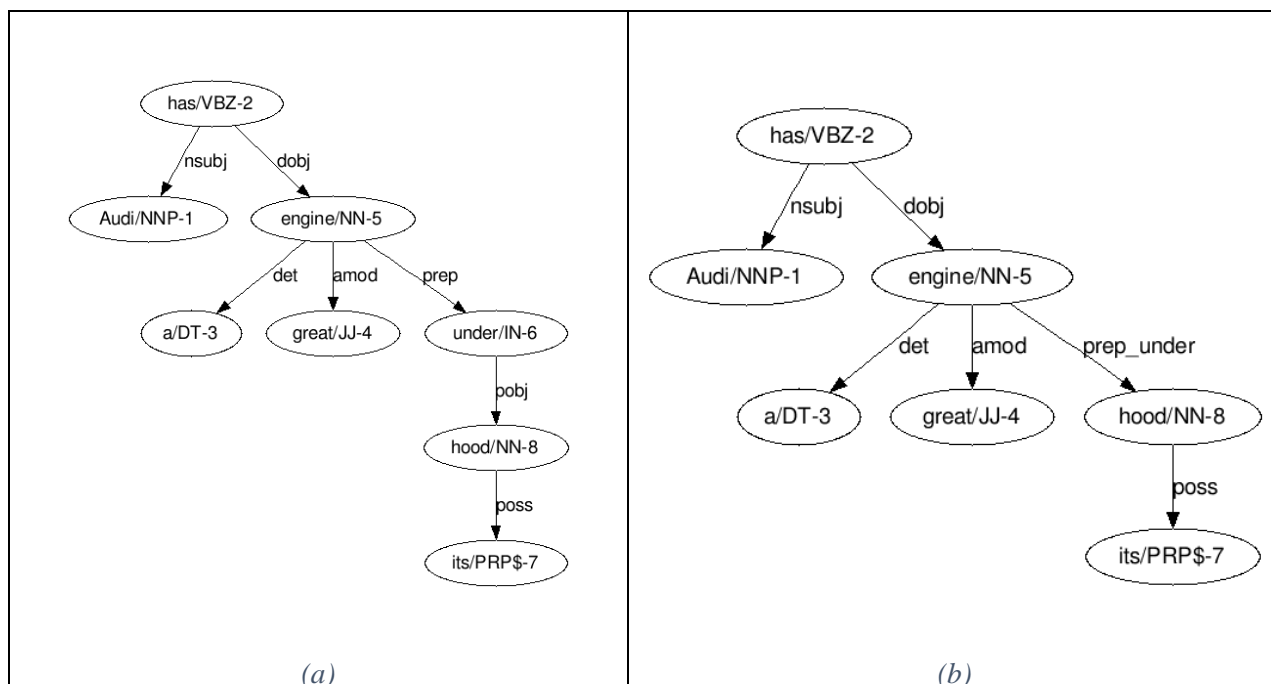
A fenti függőségeket gráfként tárolják. A 2. Ábra a fenti függőségeknek megfelelő gráfot mutatja:



2. 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” szókapcsolatban áll egyszerre az „engine” és a „hood” szavakkal. A szövegelemző felismerve ezt mosósítja a kapcsolat típusát 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. Ábra az alap- és összevont függőségeket mutatja.



3. 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.Contains(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
    endfor
    return relations
end

```

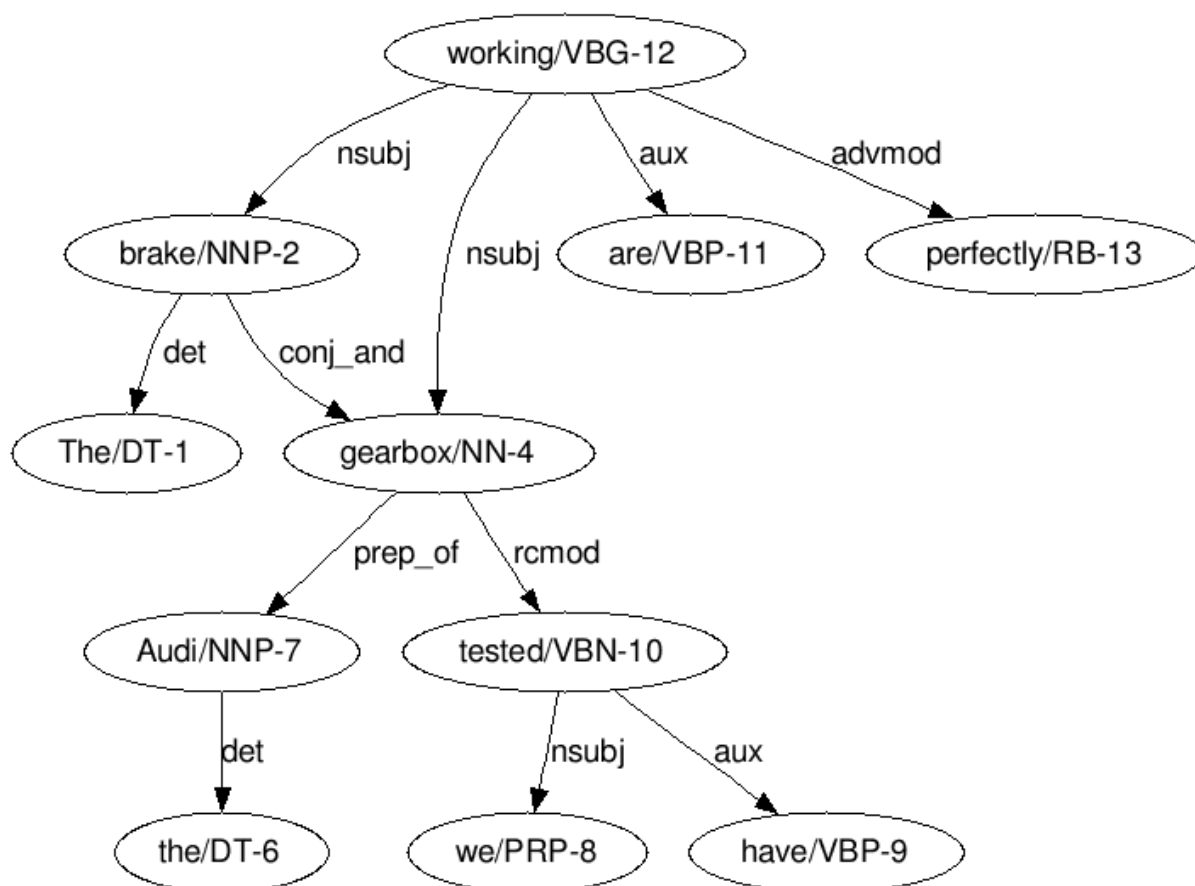
Az előbbi pszeudokódban a következő adatstruktúrákat és jelöléseket használtam:

6. Táblázat Pszeudokód magyarázata

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 4. Ábra mutatja a mondat összevont függőségi gráfját.



4. 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 a 7 Táblázat foglalja össze.

7. Táblázat Távolság mátrix

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

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.

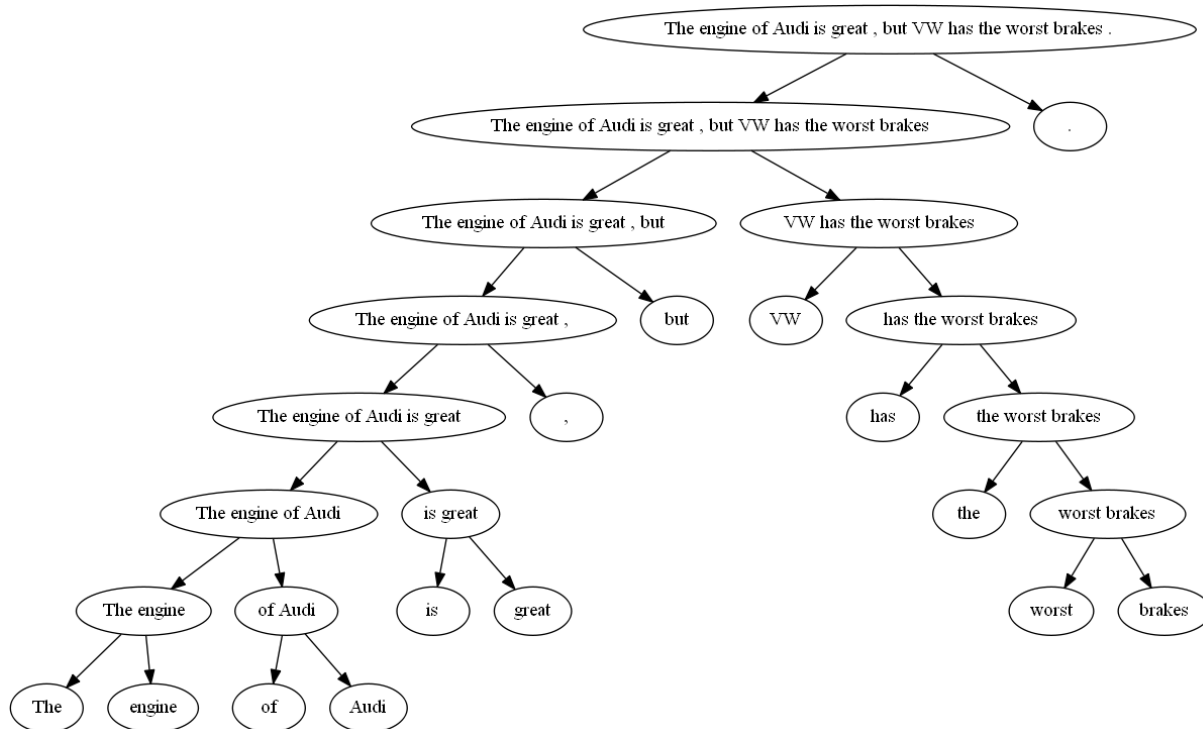
Utolsó lépésként az algoritmus megvizsgálja, hogy létezik-e a „gearbox”-tól más tulajdonsághoz vezető út, melynek hossza megegyezik 1-el (az „Audi” és „gearbox” közti távolsággal). Ebben az esetben létezik ez az út, így az algoritmus megtalálta az „Audi” és „brake” közt fennálló kapcsolatot is. Láthatjuk, hogy a fenti példa esetében az algoritmus a mondatban lévő mindkét kapcsolatot megtalálta. Az 4.4.3. fejezetben az eredmények bemutatásánál, részletesen kitérek majd az algoritmus hatékonyságára, melyet ún. „unit tesztek” segítségével, a fent említett minden esetre, több példamondaton keresztül teszteltem.

3.5 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, negatívát a negatívaknak, majd összesítik a pontokat [14]. 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 [15]. 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 érzelelemzésnél. 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. Az 5. 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.



5. Ábra Érzelemi fa

A 5. 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.

8. Táblázat Érzelem értékek

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%

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.Contains(r.Characteristic) AND node.HasVerb()
                sentiments.Add(r, node.SentimentValues())
            endif
        endfor
    endfor

```

```
endfor
return sentiments
end
```

4 A rendszer megvalósítása

4.1 A rendszer specifikációi és architektúrája

Az általam elkészített rendszer funkcionális moduljai, melyek a 6 ábrán láthatók, a következők:

- 1) Adminisztrációs modul
- 2) Tweet-gyűjtő modul
- 3) Szövegelemző modul
- 4) Érzelelemző modul
- 5) Eredményjelző modul

Az 1) adminisztrációs modul tartalmazza keresési beállítások, szövegelemző beállítások és szerviz üzemeltetését szolgáló almodulokat. A keresési beállításokon keresztül a felhasználó megjelenítheti listák formájában az általa használt kategóriákat, kulcs-szavakat és szinonimákat. A kategóriák listába vihet be új elemeket, módosíthat és törölhet létező elemeket, a kulcs-szavak és szinonimák listájába vihet be új elemeket és törölhet létezőket. A szövegelemző beállításokon keresztül a felhasználó megjelenítheti a kategóriákon belül kielemezendő tulajdonságok listáját, vihet be új elemeket és törölhet létezőket. A szerviz üzemeltetése almodul az adatbázisban eltárolt flag-ek értékének a változtatására szolgáló felhasználói felület. A felhasználó ezek segítségével indítja el és állítja meg a szervizek üzemeltetését.

A 2) Tweet-gyűjtő modul szerepe a Twitter-ről letölteni azokat a tweeteket, amelyek megfelelnek a keresési kritériumoknak és ezeknek az elmentése saját adatbázisba.

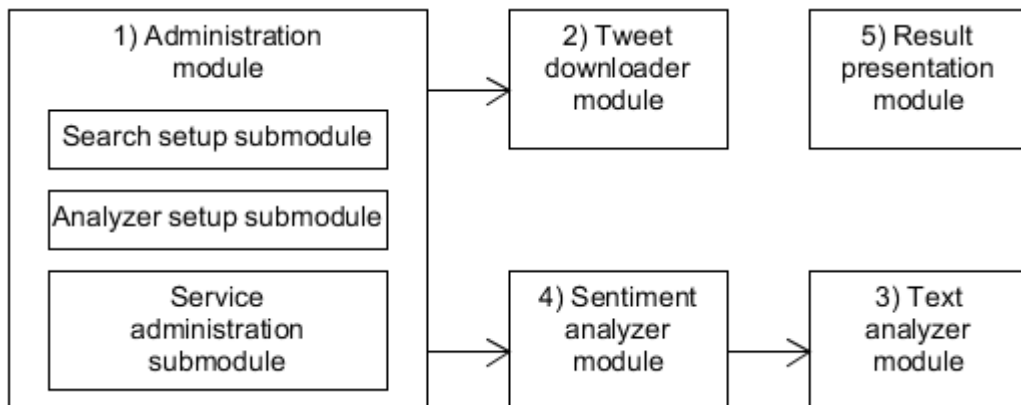
A 3) szövegelemző modul szerepe, mondattani és reláció-függőségi elemzés segítségével, a tweet szövegében megkeresni a kapcsolatokat a kulcsszavak és tulajdonságok között.

A 4) Tweet-érzelelemző modul szerepe a letöltött tweetek szövegének a feldolgozása és érzelem-elemzése.

Az 5) eredményjelző modul szerepe a felhasználó számára megjeleníteni az 4) –es érzelelemző modul által kiértékelt eredményeket.

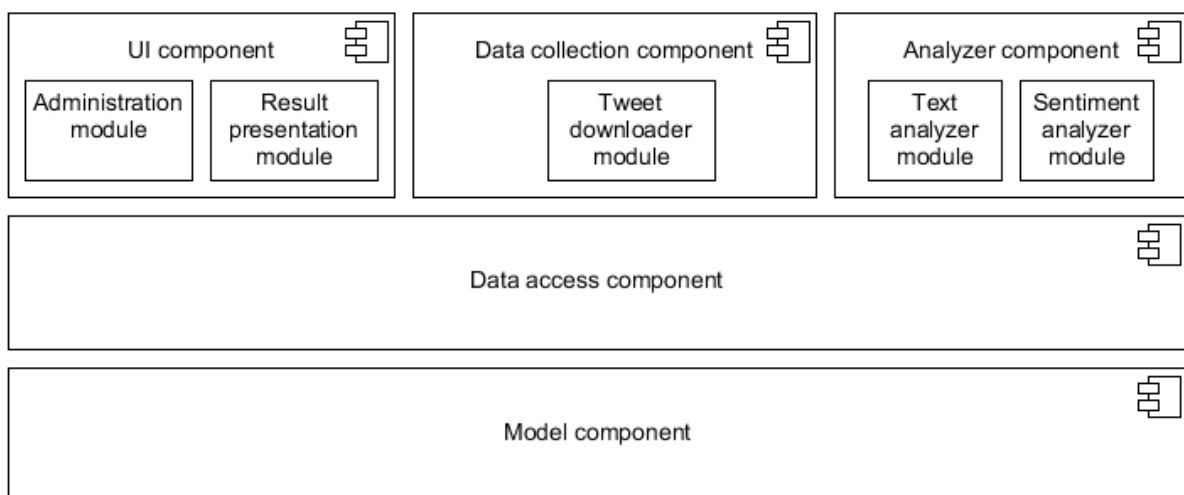
Az általam elkészített rendszer tartalmaz egy web applikációt, két szerviz komponenst, egy Data access komponenst és egy modell komponenst. A web applikáción keresztül éri el a felhasználó az adminisztrációs és az eredményjelző modult. A két szerviz lehetőséget ad a tweetek letöltésének, valamint az érzelem elemzés elindítására és megállítására. A Data access

komponensben lévő osztályok segítségével olvassuk ki és módosítjuk az adatbázisban tárolt adatokat.



6. Ábra Rendszer funkcionális architektúra

A modell komponens tartalmazza az adatbázisban szereplő tábláknak osztályokra való leképezését (ORM - object relational mapping).



7. Ábra Rendszer architektúra

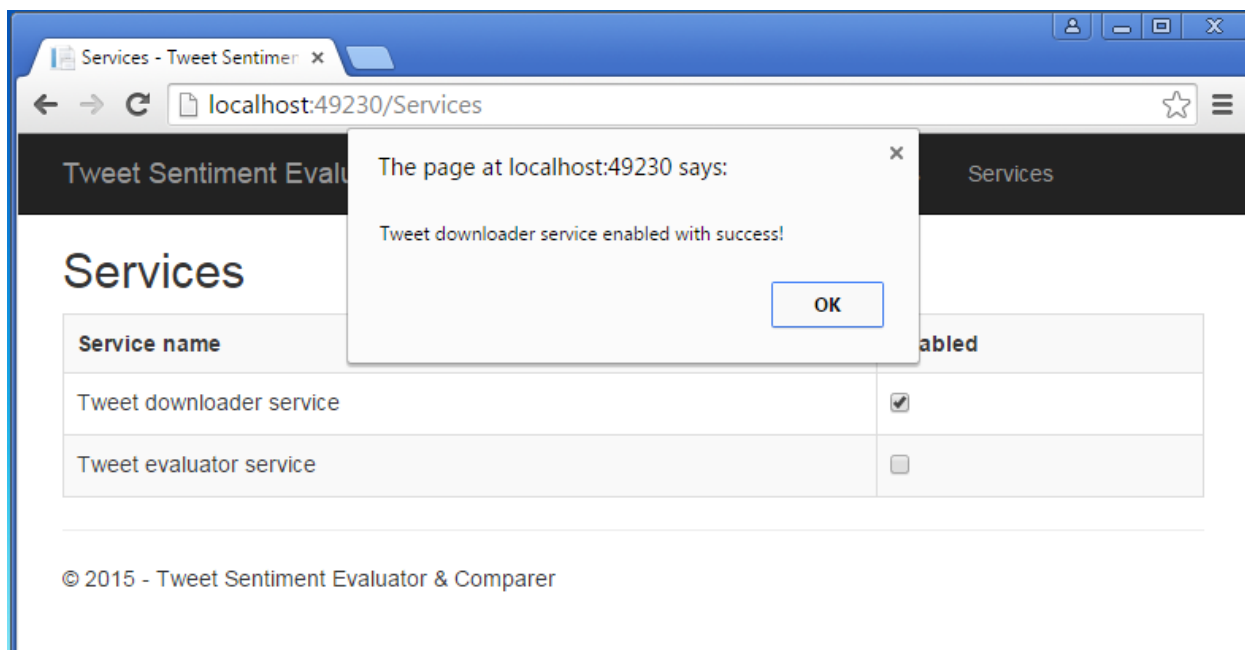
4.2 A részletes tervezés

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

4.2.1 UI komponens

Az UI komponens tartalmazza a 6 ábrán bemutatott 1) –es adminisztrációs és 5) –ös eredményjelző modulokat.

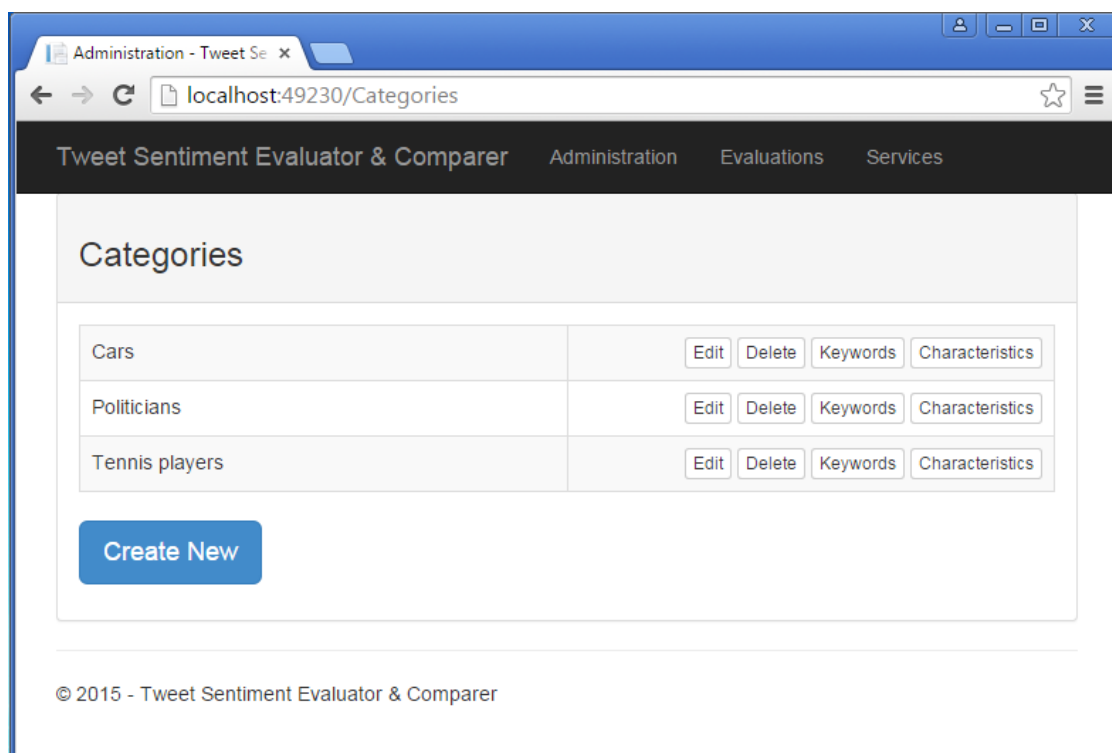
Az adminisztrációs komponens egy web applikációval valósítottam meg. A felhasználói felület megjelenítéséhez használtam a Twitter Bootstrap által kínált lehetőségeket.



8. Ábra Szervizek üzemeltetése

Az adminisztrációs komponensnek a felhasználói felületei a 8 - 15 ábrákon láthatók.

Az adminisztrációs komponens, a 6 ábrán 5) -ös eredményjelző modulja, két fő részre osztható a következő szerepekkel: a kiértékelési feltételek beállítása és az eredmények megjelenítése.



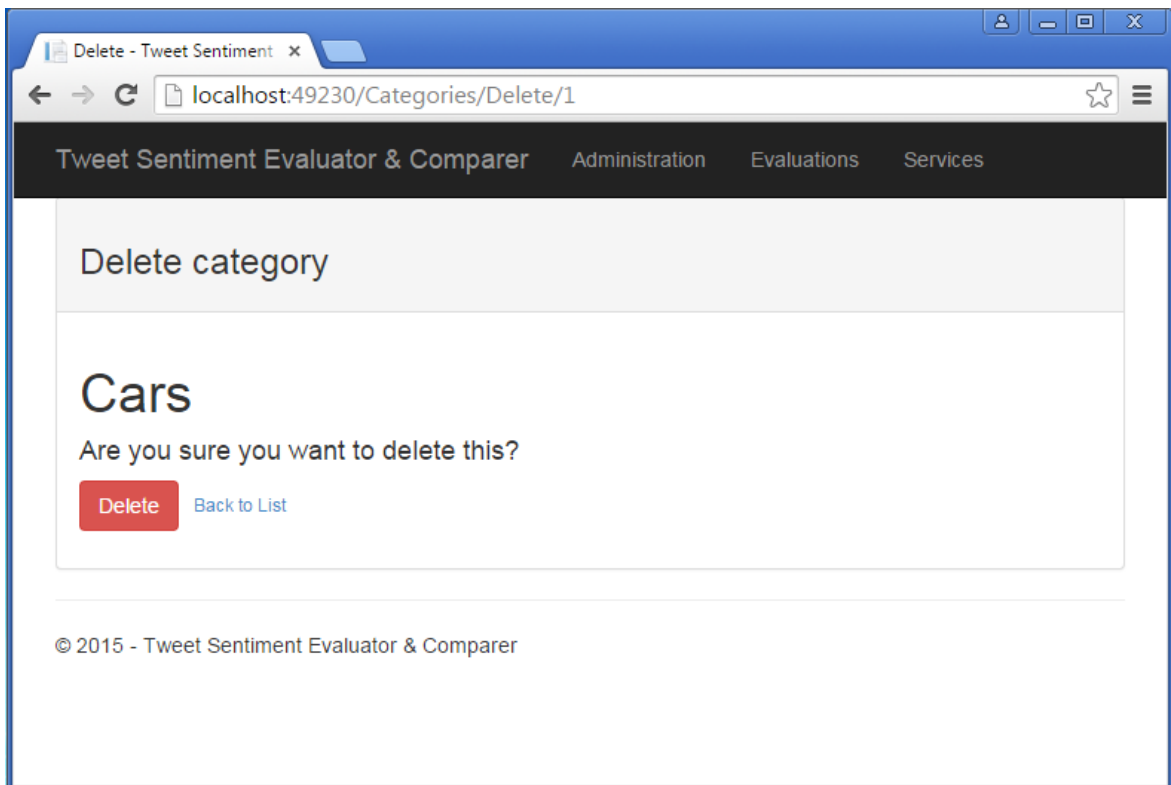
9. Ábra Kategóriák lista

The screenshot shows a web browser window with the title 'Create - Tweet Sentiment'. The address bar displays 'localhost:49230/Categories/Create'. The application's navigation bar includes 'Tweet Sentiment Evaluator & Comparer', 'Administration', 'Evaluations', and 'Services'. The main content area is titled 'Create category' and contains a form with a 'Category name' label and an empty text input field. Below the input field is a blue 'Create' button and a blue link labeled 'Back to List'. The footer of the page shows the copyright notice '© 2015 - Tweet Sentiment Evaluator & Comparer'.

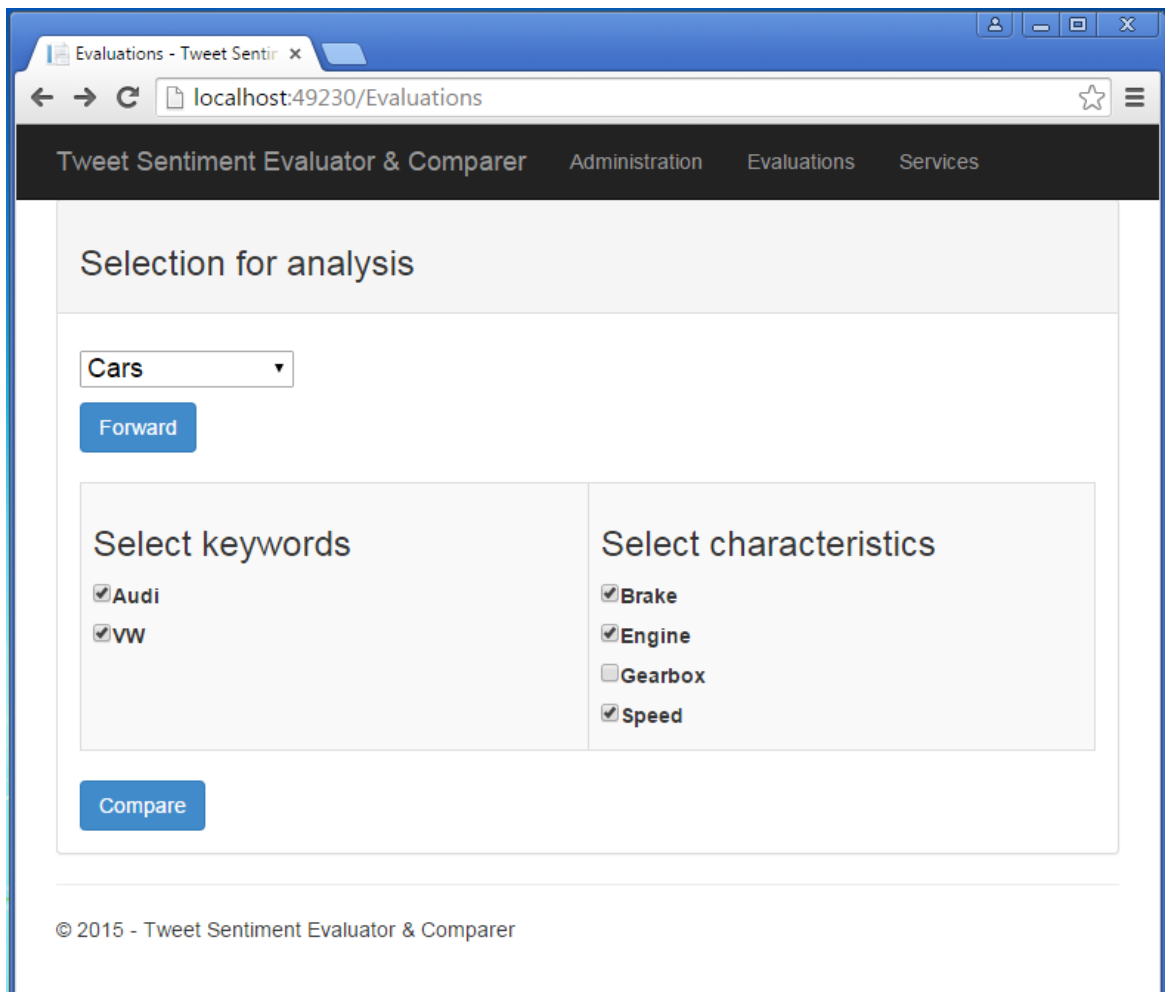
10. Ábra Új kategória létrehozása

The screenshot shows a web browser window with the title 'Edit - Tweet Sentiment Ev.'. The address bar displays 'localhost:49230/Categories/Edit/1'. The application's navigation bar includes 'Tweet Sentiment Evaluator & Comparer', 'Administration', 'Evaluations', and 'Services'. The main content area is titled 'Edit category' and contains a form with a 'Category name' label and a text input field containing the word 'Cars'. Below the input field is a blue 'Save' button and a blue link labeled 'Back to List'. The footer of the page shows the copyright notice '© 2015 - Tweet Sentiment Evaluator & Comparer'.

11. Ábra Kategória módosítása



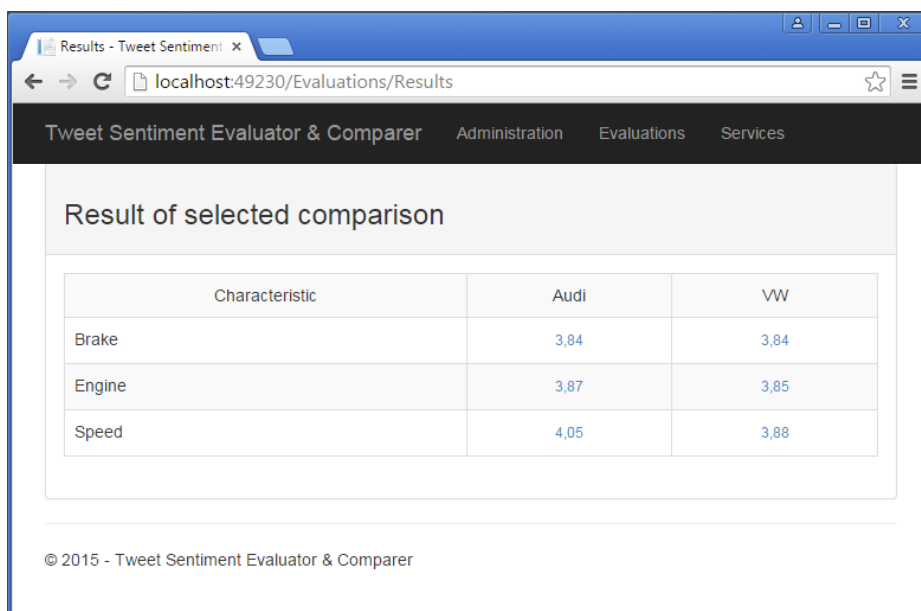
12. Ábra Kategória törlése



13. Ábra Érzelelemzési eredmények szűrése

A feltételek meghatározásánál a felhasználó elsősorban kiválaszt egy kategóriát, majd a kategórián belül létező kulcsszavak és tulajdonságok közül választhat több elemet is az összehasonlítás elvégzéséhez.

Az összehasonlítás eredménye egy táblázatként jelenik meg. A táblázat egyetlen jeggyel lássa el a kulcsszó – tulajdonság párokat. Ez tekinthető meg a 14 ábrán.



The screenshot shows a web browser window with the URL 'localhost:49230/Evaluations/Results'. The page title is 'Results - Tweet Sentiment'. The main heading is 'Result of selected comparison'. Below it is a table comparing Audi and VW across three characteristics: Brake, Engine, and Speed. The scores are: Brake (Audi: 3.84, VW: 3.84), Engine (Audi: 3.87, VW: 3.85), and Speed (Audi: 4.05, VW: 3.88). The footer indicates '© 2015 - Tweet Sentiment Evaluator & Comparer'.

Characteristic	Audi	VW
Brake	3.84	3.84
Engine	3.87	3.85
Speed	4.05	3.88

14. Ábra Érzelelemzési eredmények

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



The screenshot shows a modal window titled 'Related tweets and sentiment' overlaid on the main application. The modal contains a table with five columns: Text, Very negative, Negative, Neutral, Positive, and Very positive. It lists five tweets related to the comparison, along with their sentiment scores. The background shows the same comparison table as in the previous screenshot.

Text	Very negative	Negative	Neutral	Positive	Very positive
@hpbldv nope... the prosche engine was from about '62, just a little more powerful than the VW engine... same size, same basic look too	8,00	33,00	17,00	30,00	9,00
1987 VW Vanagon Westfalia Camper w/ 10k on GoWesty Engine - \$12,500 in Indiana http://t.co/0KPEV1PLPn http://t.co/Av4nxMTLV3	14,00	65,00	16,00	2,00	1,00
RT @WestysForSale: 1987 VW Vanagon Westfalia Camper w/ 10k on GoWesty Engine - \$12,500 in Indiana http://t.co/0KPEV1PLPn http://t.co/Av4nxMTLV3	12,00	60,00	19,00	4,00	2,00
VW RABBIT GOLF JETTA SCIROCCO	16,00	64,00	14,00	2,00	1,00

15. Ábra Érzelemeredményhez tartozó tweetek

A 15 ábrán látható a Stanford Parser érzelemelemző által kiértékelt eredmények a tweet szövegére. Itt megjegyezném, hogy csak arra a szövegrészre hajtódott végre az érzelemelemzés, amelyet a rendszer készített elő az érzelemelemző modult felhasználva. Az elemző komponens, mely tartalmazza a 6 ábrán bemutatott 4) -es érzelemelemző modult, részletesen a 4.2.3 fejezetben mutattam be.

4.2.2 Adat-gyűjtő komponens

A microblogging napjainkban egy nagyon népszerű kommunikációs eszközzé vált az internetezők között. [16] Felhasználók 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 Twittert érdemes elemezni a következők:

- különböző személyek különböző topikokról nyilvánítanak ki véleményeket, tehát egy értékes vélemény-forrást képez; az adathalmaz, amit tartalmaz óriási és naponta növekedik;
- a twitter felhasználó típusai változatosak, átlagos felhasználóktól celebekig, cégek, politikusok, még államelnökök is, tehát változatos csoportok véleményével találkozunk különböző szociális és érdekképviselői csoportoktól;
- a felhasználók különböző országokból vannak.

A twitter.com egy online szociális hálózat melyet milliók használnak a föld különböző országaiból kapcsolat-tartásra családtagokkal, barátokkal, munkatársakkal számítógépen vagy mobil-eszközökön keresztül. [17] A felhasználói felület rövid szöveges üzenetek posztolását teszi lehetővé (140 karakter maximum hosszal) amelyek láthatóak bármely más Twitter felhasználó számára. A felhasználók, megjelölhetik azokat a személyeket, akiket szeretnének követni, ez esetben a felhasználó értesítve lesz mikor a követett új üzenetet posztolt. A követettnek jelölt felhasználó nem feltétlenül kell vissza- kövesse azt a személyt aki őt követi, így módon a twitter egy irányított kapcsolatú szociális hálót alkot. [18]

A tweet az elemi parányi építő-tömbje a Twitternek. Felépítését a 9 táblázatban mutatom be:

9. Táblázat Tweet felépítése

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 szövegéből lettek kiszűrve, mint pl.hashtagek, url-k.
favorite_count	Integer	Nullázható. Hány felhasználó jelölte meg kedvencként a tweetet.
favorited	Boolean	Nullázható. A hitelesítő 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 azonosítója, egész számként.
id_str	String	A tweet egyedi azonosítója, 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 hozzáfűződő 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 hitelesített 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 szövege.
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.

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

A REST API biztosítja a programokon keresztüli hozzáférést, olvasást és írást, a Twitter adataihoz. Létrehozhatunk új Tweet-et, olvashatjuk, a felhasználók profilját és a követők adatait, egyedi kereséseket hajthatunk végre és sok minden egyebet. A REST API a felhasználók és applikációk 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 relevancia 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 azonosítást használva, egy adott applikáció 450 lekérdezés/kérés 15 percnyi időtartamra a határ, applikáción 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 háromtípusú 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 folyamatok – A több-felhasználó változata a felhasználó folyamatoknak. Az oldal folyamatok azokat a szervereket célozzák, amelyek a több felhasználó nevében kell kapcsolódjanak a Twitterhez.

A 10 táblázat bemutatja az adatbázisban szereplő táblákat, melyekbe az adatokat mentjük.

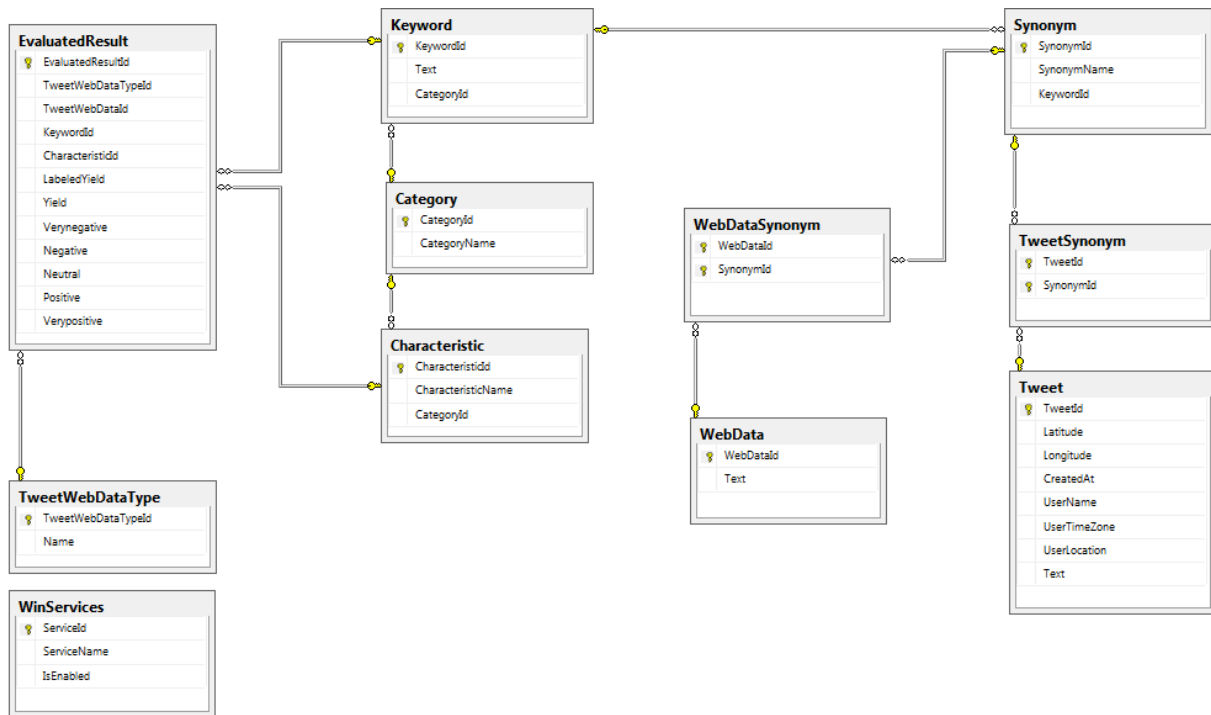
10. Táblázat Adatbázisban szereplő táblák listája

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

A 16 ábrán látható az adatbázis egyed-kapcsolat diagramja. Az adat-gyűjtő komponens tartalmazza a 6 ábrán bemutatott 2) –es tweet-gyűjtő modult. A komponens szervizként működik az applikáción belül és az adminisztrációs felületről indítható el illetve állítható meg.

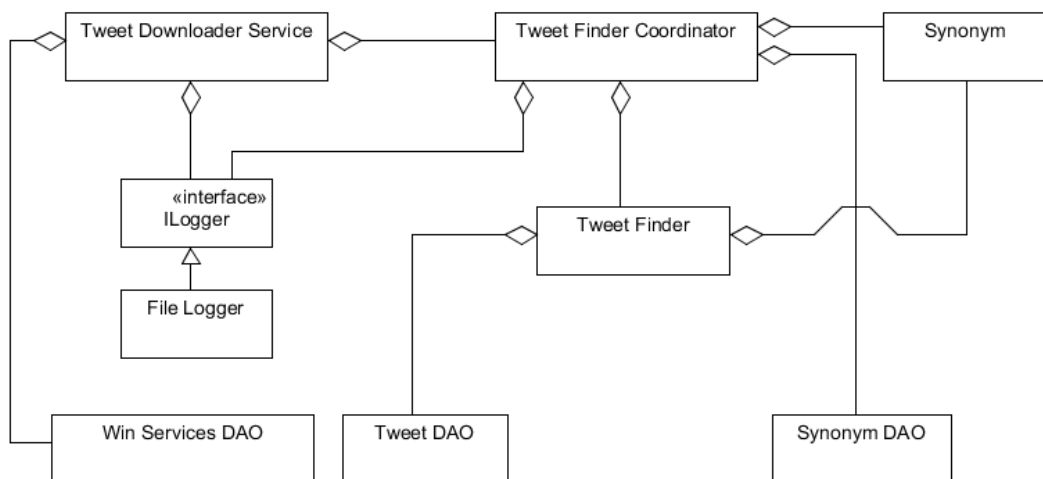
Az adat-gyűjtés komponens osztályai a következők (lásd 17. Ábra):

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.



16. Ábra Egyed-kapcsolat diagram

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 `_isEnabledInDatabase` adattag értéke tükrözi az adatbázisban szereplő érzelemelemző folyamat státuszát, futhat az érzelemelemző szervíz vagy nem.

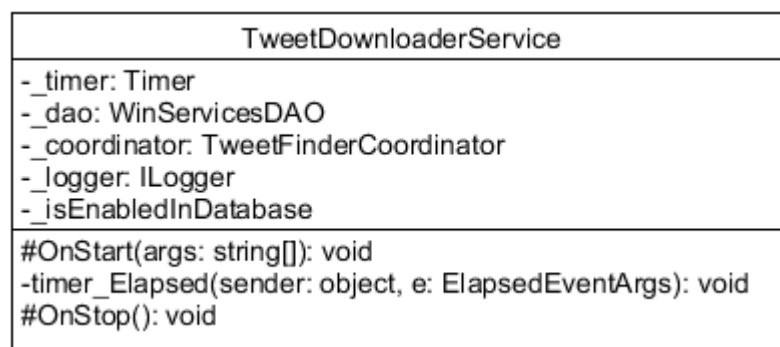


17. Ábra TweetDownloader osztály diagram kapcsolatok

Az `OnStart` metódus inicializálja a `_timer`, `_dao`, `_logger` és `_coordinator` adattagokat és a `_timer` időzítő segítségével meghívja a beállított időközönként a `timer_Elapsed` metódust. A `timer_Elapsed` metódus 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 naplózza a kapott beállításnak az állapotát és a `_coordinator` adattagon keresztül elindítja vagy megállítja a tweet gyűjtési folyamatot. Az `OnStop` metódus a `_coordinator` adattagon keresztül leállítja a tweet gyűjtési folyamatot.

A szerviz működési folyamata a következő: létrehozza a kapcsolatot az adatbázissal, utána elindít egy időzítőt, amely 10 másodpercenként ellenőrzi az adatbázisban, hogy a letöltési szerviz engedélyezve van-e vagy nem, és ha ennek az állapota megváltozott az előző állapotához képest. Ha nem volt változás, akkor 10 másodperc múlva újraindul az adatbázisos ellenőrzés, ha változás volt, akkor a letöltési folyamatengedélyezési állapot szerint elindítja a letöltést vagy megállítja.

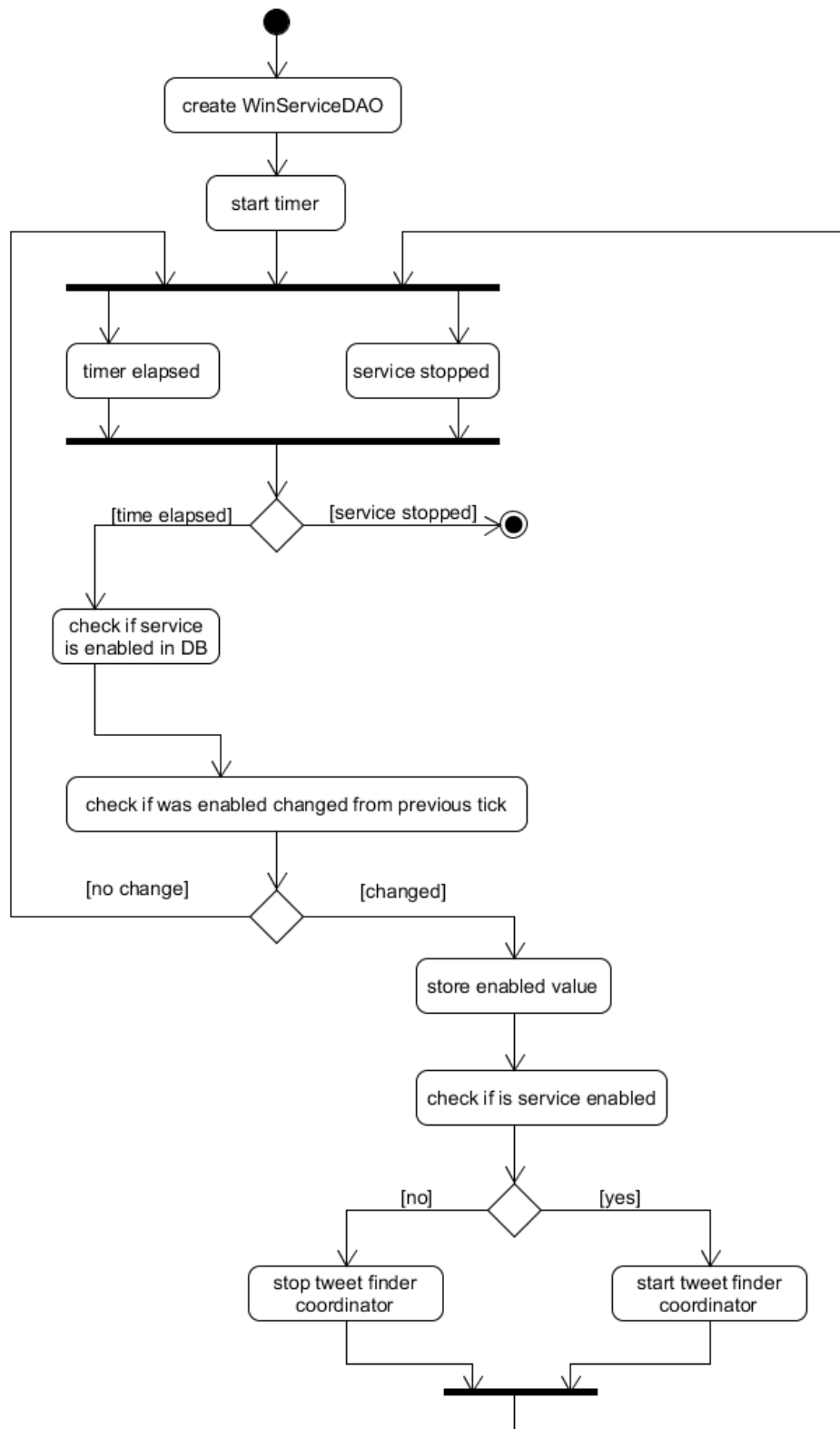


18. Ábra `TweetDownloaderService` osztály diagram

TweetFinderCoordinator szerepe a tweetek keresési és lementési folyamatának a vezérlése. A `_tweetFinder` adattagon keresztül fogjuk elérni a `TweetFinder` osztály metódusait. 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.

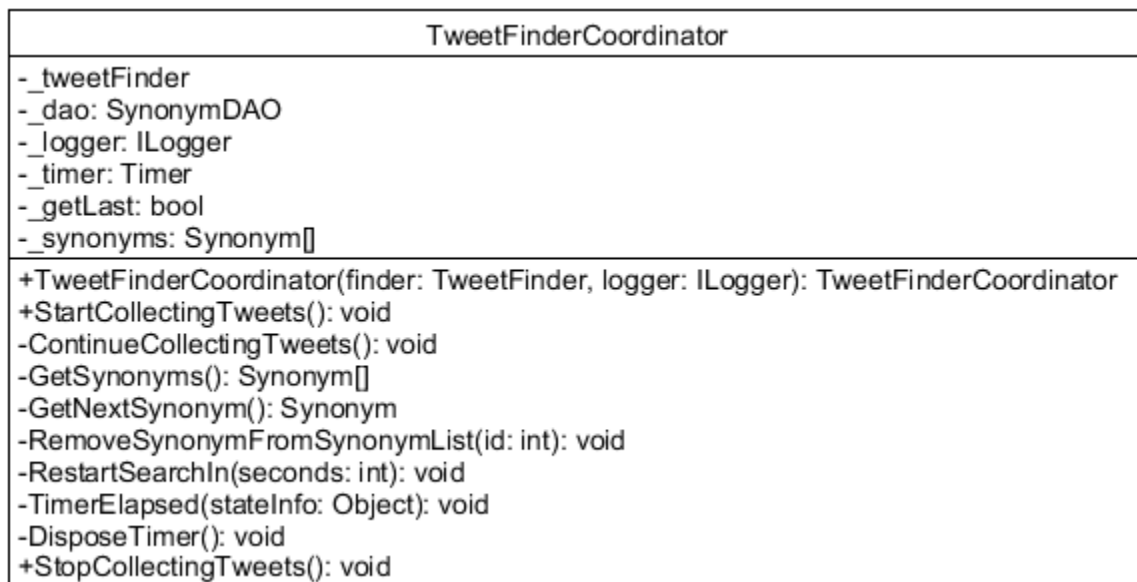
A `TweetFinderCoordinator` osztály konstruktorában inicializáljuk a `_dao` adattagot valamint injektáljuk a `_logger` és a `_tweetFinder` adattagokat. A `StartCollectingTweets` meghívja a `StopCollectingTweets` metódust, feltölti a `_synonyms` adattagot a `GetSynonyms` metódus meghívásával majd meghívja a `ContinueCollectingTweets` metódust. A `ContinueCollectingTweets` metódus a `GetNextSynonym` metódussal lekérdezi egy lokális változóba a következő keresett szinonimát. Ha a változó értéke null, akkor naplózza az információt, hogy nem talált keresésre szánt szinonimát, és meghívja a `RestartSearchIn` metódust a 60 értékű paraméterrel. Ha nem null a keresett szinonima értéke, akkor naplózza az információt, hogy a keresést elindította az adott szinonimára és meghívja a `_tweetFinder` adattagról a `StartSearchAsync` metódust átadva a szinonima értékét paraméterként. A

GetSynonyms metódus a _dao adattagnak megpróbálja meghívni a GetSynonymsOrderedByTweetCount metódusát, ha ez sikerült, akkor visszatéríti a szinonimák listáját, ha nem sikerül, akkor naplózza az információt, hogy a szinonima lista betöltése nem sikerült és null értéket térít vissza.



19. Ábra TweetDownloaderService activity diagram

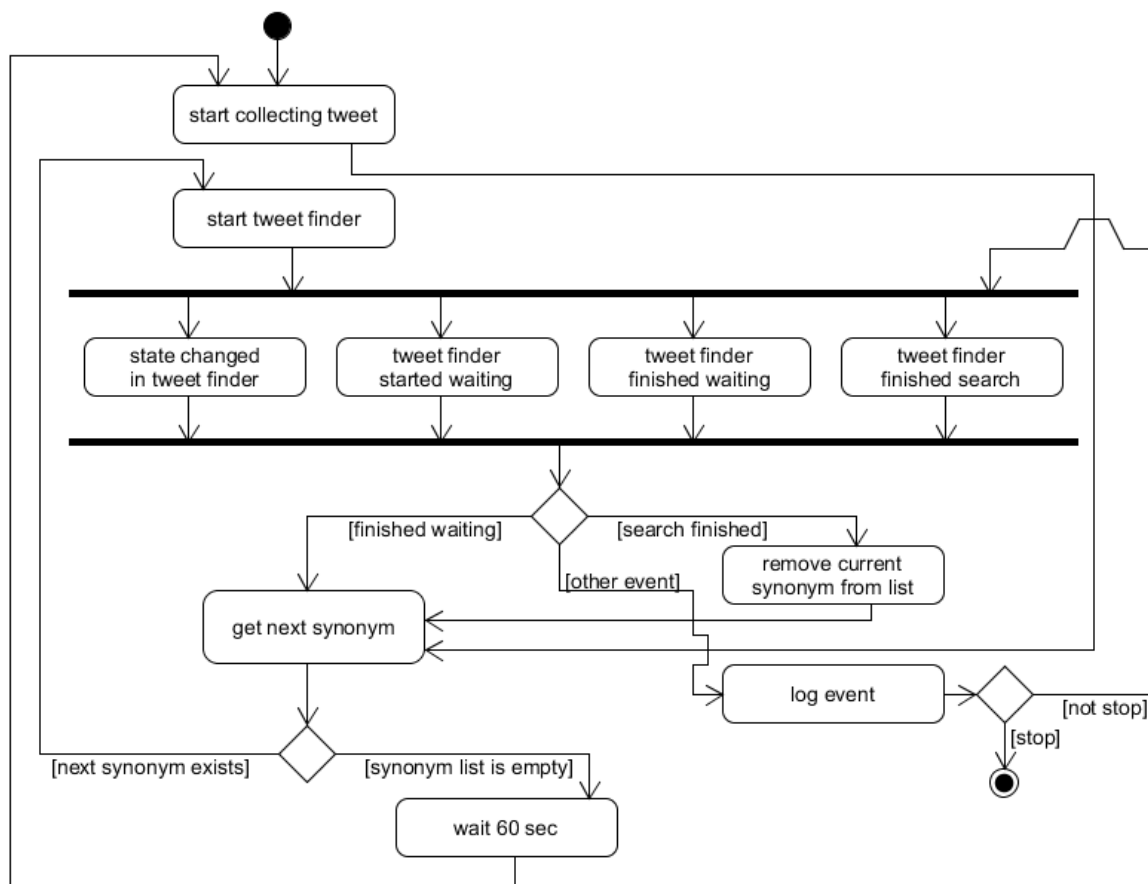
A `GetNextSynonym` metódus abban az esetben, ha a `_synonyms` lista üres null értéket térít vissza. Ha a lista nem üres, akkor a `_getLast` adattag értéke függvényében, ha igaz, akkor az utolsó elemet, ha hamis, akkor az első elemet téríti vissza. A `RemoveSynonymFromSynonymsList` paraméterként kapott szinonim azonosító értéket megkeresi a `_synonyms` listában. Ha megtalálta és a `_synonyms` lista nem üres, akkor törli a listából. A `RestartSearchIn` metódus inicializálja a megkapott paraméter értékével a `_timer` adattagot. A `TimerElapsed` metódus meghívja a `DisposeTimer` metódust, lenullázza a `_synonyms` adattag értékét majd meghívja a `StartCollectingTweet` metódust. A `DisposeTimer` metódus felszabadítja a `_timer` adattag által lefoglalt erőforrásokat és az értékét nullára állítja. A `StopCollectingTweets` metódus szintén meghívja a `DisposeTimer` metódust és a `_tweetFinder` adattagról a `StopSearch` metódust.



20. Ábra `TweetFinderCoordinator` osztály diagram

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 befejezte a keresést. Ha a kereső befejezte a várakozást, akkor a folyamat a következő szinonim lekérdezésével folytatódik. A keresés befejezése után eltávolítjuk a szinonima listájából a keresett szinonimát és folytatjuk a következő szinonima lekérdezésével. Ha egyéb esemény következik be, akkor naplózzuk az eseményt és megállítsuk vagy újakezdjük 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 azonosítót. 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.



21. Ábra TweetFinderCoordinator activity diagram

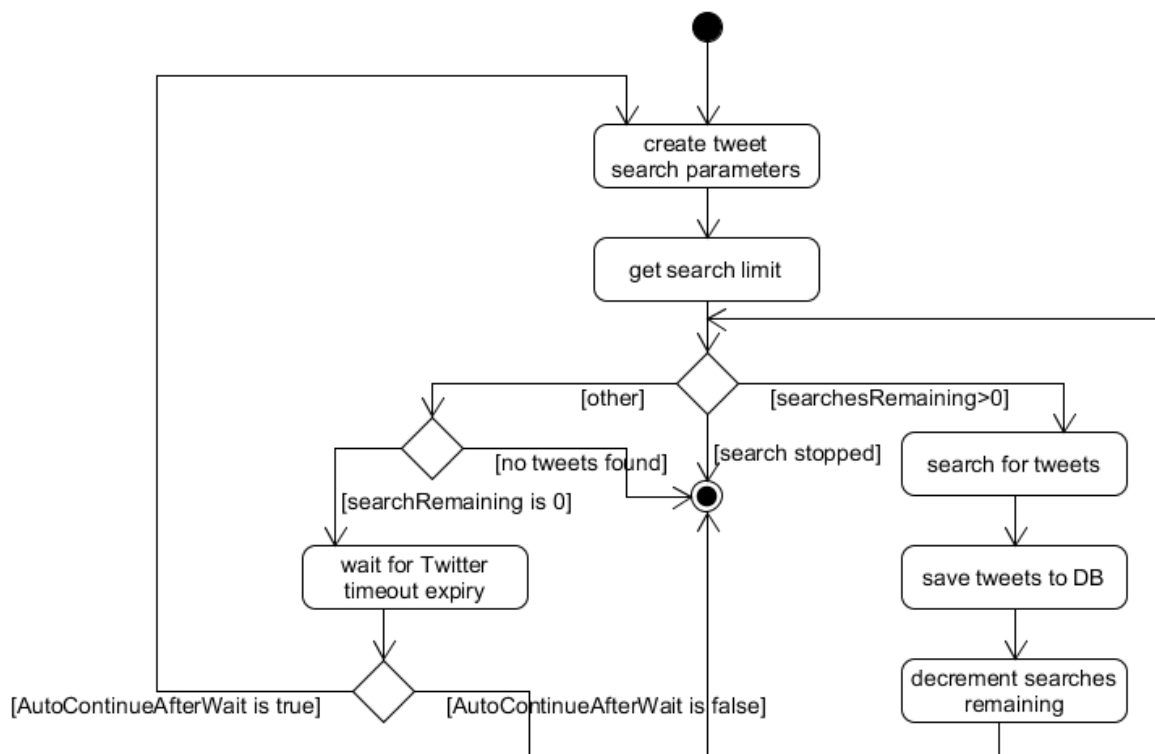
A `StartSearchAsync` metódus megkapja paraméterként a keresett szinonimát, meghívja a `StopSearch` metódust, átadja a paraméter értékét a `_synonym` adattagnak, igaz értéket ad az `IsSearchInProgress` adattagnak, hamis értéket ad `_searchStopped` adattagnak, zéró értéket ad a `_totalTweetsSaved` adattagnak, meghívja a `DisposeWaitTimer` metódust, jelzi az `OnSearchStarted` esemény bekövetkezett és meghívja a `ContinueSearch` metódust. A `ContinueSearchAsync` metódus előállítja a keresési paramétereket a `CreateTweetSearchParameters` metódus meghívásával, lekérdezi a Tweet API korlátozásait a

GetSearchTweetsLimit metódussal, beállítva a searchesRemaining változó értékét a megengedett letölthető tweetek számára, 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 zéró, akkor a TweetInvi csomagot használva beállítsuk twitter API-hoz szükséges azonosítónkat és meghívjuk a tweet API által biztosított keresési metódust. A letöltött tweetek egy ITweet típusú elemekből álló listába kerülnek. Csökkentjük a searchesRemaining változó értékét és naplózzuk a letöltött tweetek számát valamint a searchesRemaining értékét. Ha a hiba lép fel, akkor lenullázzuk a tweets listát, zéró értéket adunk a searchesRemaining változónak és naplózzuk a hiba jellegét. Ha a tweets listában léteznek elemek, akkor meghívjuk a SaveTweets metódust, frissítjük a _totalTweetsSaved adattagot és naplózzuk az adatbázisba sikeresen lementett tweetek számát. Ez a keresés és letöltési folyamat mindaddig ismétlődik, míg a searchesRemaining változó értéke nagyobb mint zéró és a _searchStopped adattag értéke hamis és a letöltött tweet listában vannak elemek. Ha a searchesRemaining változó értéke zéró lesz, akkor naplózzuk, hogy az idő lejárt és meghívjuk a SetupWaitTimer metódust átadva paraméterként a várakoztatási idő értékét. Ha a _searchStopped értéke igaz lesz, akkor _isSearchInProgress adattagnak az értékét hamisra állítjuk és naplózzuk, 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, naplózzuk az eseményeket, hogy több tweetet nem találtunk és összesen hány tweetet mentettünk le.

TweetFinder
- _synonym: Synonym - _dao: TweetDAO - _credentials: IOAuthCredentials - _searchStopped: bool - _waitTimer: Timer - _totalTweetsSaved: int +SearchKeyword: Synonym +AutoContinueAfterWait: bool - _isSearchInProgress: bool
+StartSearchAsync(synonym: Synonym): void -ContinueSearch(): void -CreateTweetSearchParameters(): ITweetSearchParameters -ExistTweetsForCurrentSynonym(): bool -FindLowestTweetIdForCurrentSynonym(): long -GetSearchTweetsLimit(): ITokenRateLimit -SaveTweets(tweets: Tweet[]): int -ConvertITweetToTweetEntity(item: ITweet): Tweet -SetupWaitTimer(periodToWait: TimeSpan): void -WaitTimerCallback(stateInfo: Object): void +StopSearch(): void -DisposeWaitTimer(): void

22. Ábra TweetFinder osztály diagram

A `CreateTweetSearchParameters` eljárás készíti el a Tweet Search API – nak szükséges paraméter beállítását, 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 legkisebb 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 perzisztálja felhasználva a `TweetDAO` projekt osztályait a letöltött Tweeteket. Mivel a letöltött Tweetek típusa `ITweet` szükséges átalakítani az applikációnk által használt Tweet entitás típusá, erre használom a `ConvertITweetToTweetEntity` eljárást. A `StopSearch` eljárás megállítja a keresési folyamatot és meghív egy másik eljárást, `DisposeWaitTimer`-t, amely felszabadítja az időzítőt.



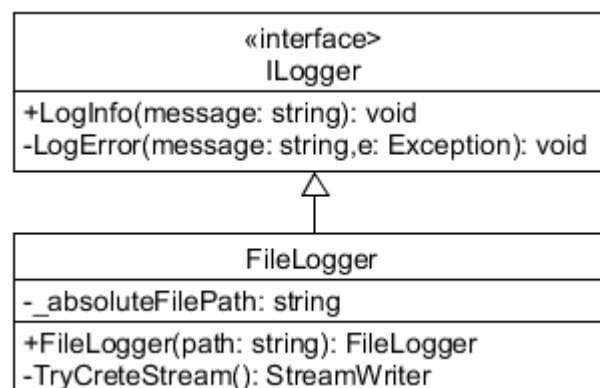
23. Ábra *TweetFinder* activity diagram

A `TweetFinder` osztály folyamata a tweet keresési paraméter létrehozásával kezdődik. Ezután lekérdezzük a Tweet API-tól a számunkra érvényes keresési korlátokat. Egy három elágazású

csomópont következik. Az első elágazás, ha a keresés leállt, ebben az esetben a folyamat leáll. A második elágazás, ha még kereshetünk tweeteket, azaz nem értük el a korlátot, akkor keresünk tweeteket, lementsük az adatbázisba, csökkentjük a maradt keresési korlátot és újakezdjük a folyamatot. A harmadik elágazás, ha egyéb, azaz a másik két eset közül egyik sem, ekkor, ha már nem kapunk több tweetet a keresést megállítjuk. Ha még nem értük el a keresési korlátot, akkor megvárjuk a korlátolt idő lejártát és annak függvényében, hogy az automatikus folytatási adattagnak igaz vagy hamis értéke van, újakezdjük a keresést vagy megállítjuk a keresést.

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

A FileLogger osztály implementálja az ILogger interfészt. Az _absoluteFilePath adattag tárolja annak a fájlnek 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 megpróbál kiírni a megkapott útvonalú fájlba, és ha sikerül, akkor visszatérít egy stream-et. Az interfész két eljárásának az implementálásában a visszatérített stream-et felhasználva naplózuk az eseményeket vagy a fellépő hibákat.



24. Ábra ILogger és Logger osztály diagram

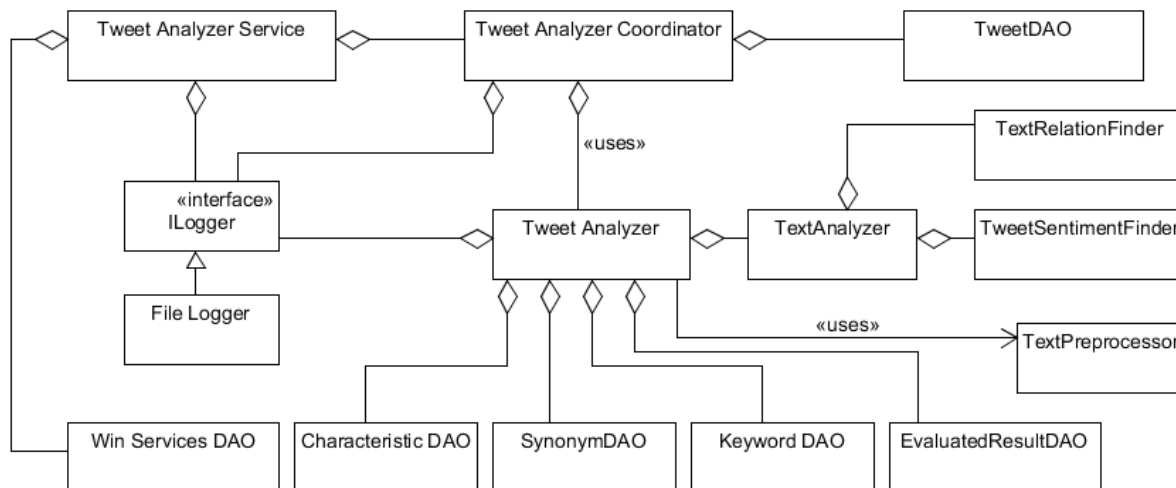
4.2.3 Elemző komponens

Az elemző komponens tartalmazza a 6 ábrán bemutatott 3) -as szövegelemző és 4) -es érzelemelemző modulokat. Ez a komponens szervizként működik az applikáción belül és az adminisztrációs felületről indítható el illetve állítható meg.

Az elemző komponens a következő osztályokat tartalmazza:

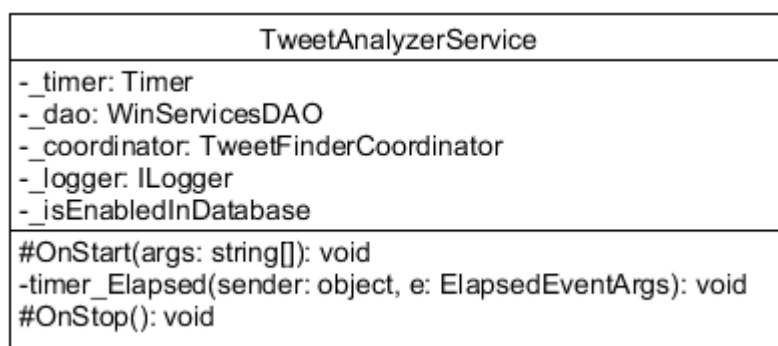
TweetAnalyzerService osztály a ServiceBase osztályt egészíti ki, szerepe a tweetek letöltési folyamatának az elindítása és megállítása. A _timer adattaggal időzítjük azt az időközt, amikor leellenőrizzük az adatbázisban a letöltés engedélyezve van-e vagy nem. A _dao adattag biztosítja a kapcsolatot az adatbázisban lévő táblával, amelyben tároljuk a tweet érzelemelemző folyamat engedélyezését. A _coordinator adattag segítségével fogjuk a tweet érzelemelemző 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 érzelemelemző folyamat státuszát, futhat az érzelemelemző szerviz vagy nem.



25. Ábra TweetAnalyzer osztály diagram kapcsolatok

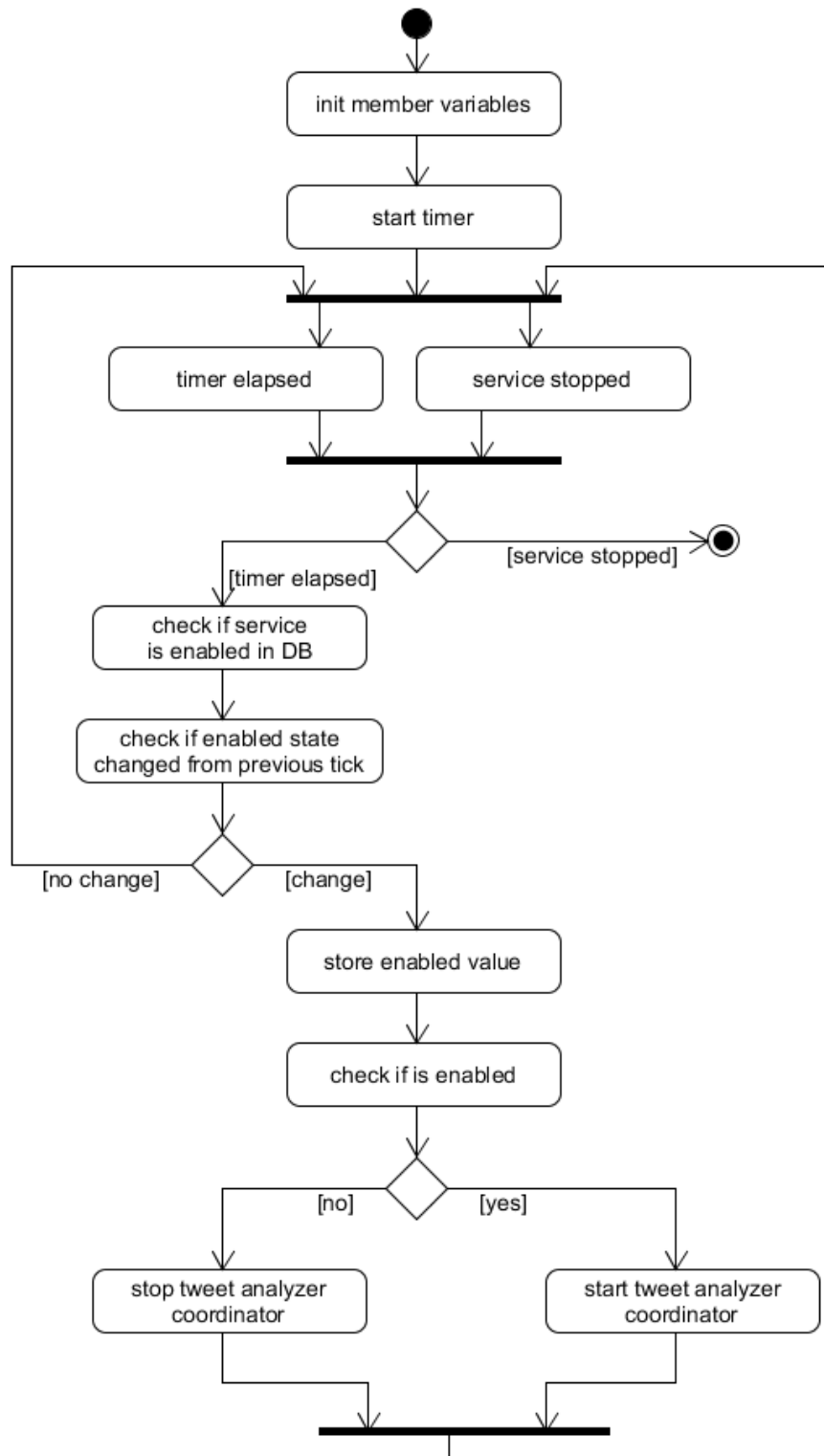
Az `OnStart` metódus inicializálja a `_timer`, `_dao`, `_logger` és `_coordinator` adattagokat és a `_timer` időzítő segítségével meghívja a beállított időközönként a `timer_Elapsed` metódust. A `timer_Elapsed` metódus a `_dao` adattagon keresztül kiolvassa az adatbázisból, hogy a tweet érzelemelemzés 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 naplózza a kapott beállításnak az állapotát és a `_coordinator` adattagon keresztül elindítja vagy megállítja a tweet érzelemelemzési folyamatot. Az `OnStop` metódus a `_coordinator` adattagon keresztül leállítja a tweet érzelemelemzési folyamatot.



26. Ábra TweetAnalyzerService osztály diagram

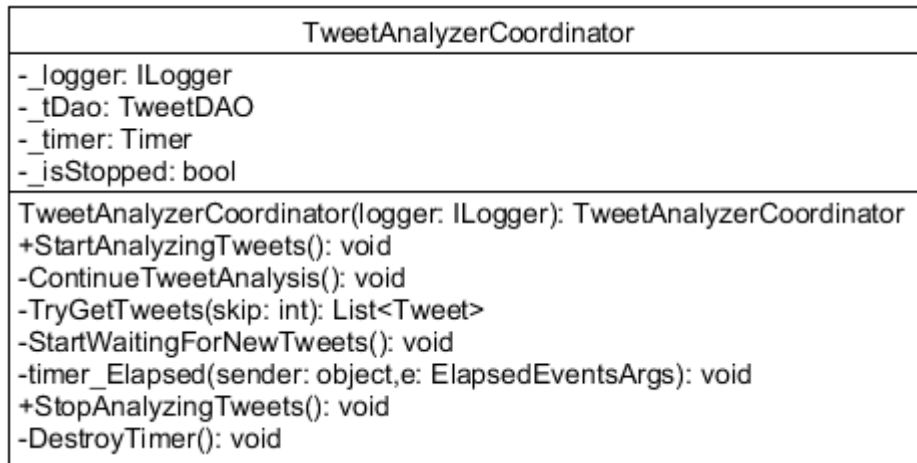
A szerviz működési folyamata a következő: inicializálja az adattagokat és elindítja az időzítőt. Ezután két eset lehetséges: lejárt az időzített periódus vagy megállították a szervizt. Ha megállították a szervizt akkor a érzelem elemző folyamat megáll. Ha az időzített periódus járt le

akkor leellenőrizzük az adatbázisban 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 lementjük az új státusz értékét. 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 szerviz folyamata újraindul mindkét esetben.



27. Ábra TweetAnalyzerService activity diagram

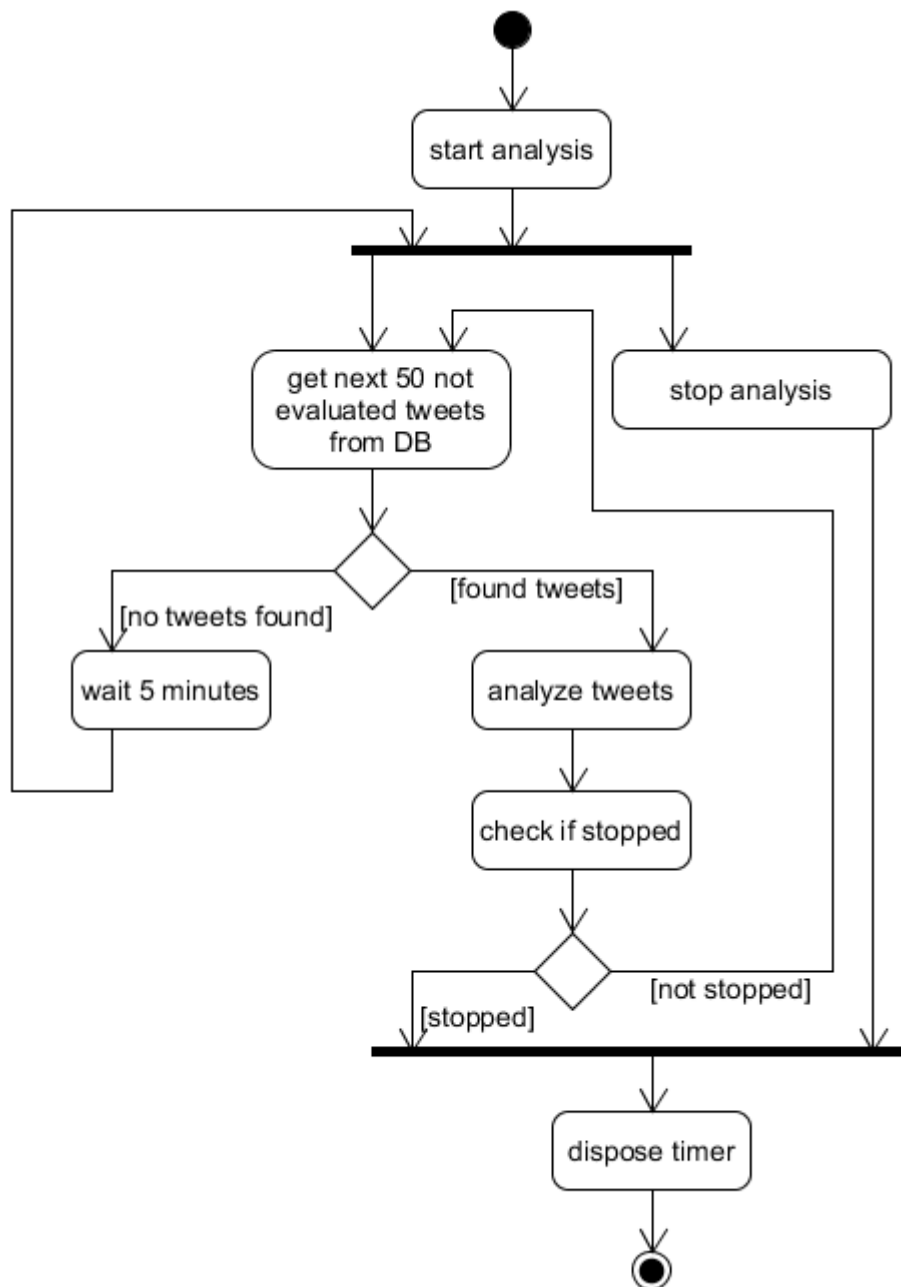
TweetAnalyzerCoordinator osztály szerepe a tweetek érzelem elemzési folyamatának a vezérlése. A `_logger` adattag segít a folyamatok naplózásában. A `_tDao` adattag segítségével hajtunk végre majd lekérdezést az adatbázis tweet táblájából. A `_timer` adattaggal időzítjük a tweet érzelemelemző folyamat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



28. Ábra TweetAnalyzerCoordinator osztály diagram

Az osztály konstruktorában beállítjuk a `_logger` adattagnak a kapott paraméter értékét és inicializáljuk a `_tDao` adattagot. A `StartAnalyzingTweets` metódus naplózza az érzelemelemző folyamat elindítását, hamisra állítja az `_isStopped` adattagot és meghívja a `ContinueTweetAnalysis` metódust. A `ContinueTweetAnalysis` metódus előkészít egy tweetekből álló listát `TryGetTweets` metódus meghívásával. Ha a lista üres, akkor meghívódik a `StartWaitingForNewTweets` metódus. Ha a listában vannak elemek, akkor inicializál egy helyi `TweetAnalyzer` típusú változót az analyzer névvel, majd egy ciklust futtat, amíg az érzelemelemzés engedélyezve van és a lista nem üres. A cikluson belül naplózzuk, hogy érzelemelemzést hajtunk végre és hány darab tweetre. A `while` cikluson belül egy újabb `while` ciklussal megyünk végig a lista elemein, míg az érzelemelemzés engedélyezve van és a lista nem üres. Ezen a második cikluson belül az analyzer-nek meghívjuk az `Analyze` metódusát, amely lementi a kiértékelt eredményeket és visszatéríti a sikeresen kiértékelt tweetek számát. Miután végigmentünk a lista elemein naplózzuk az érzelem elemzés eredményét. Ezután ha az érzelemelemző 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, meghívódik a `StartWaitingForNewTweets` metódus, más esetben naplózzuk, hogy megállt az elemzés. A `TryGetTweets` metódus paraméterként megkapja, hogy hány tweetet kell átugorjon, egy try-catch-ben meghívja a `_tDao` adattagról a

GetTweetsForEvaluation metódust. Hiba esetén naplózza a hibát és null értéket térít vissza, sikeres kapcsolódás után visszatérít egy tweetekből álló listát. A StartWaitingForNewTweets metódus naplózza, hogy tweet hiány miatt várakozás következett be, majd időzíti a várakozást 5 percre. A timer_Elapsed metódus meghívja a DestroyTimer metódust majd utána a ContinueTweetAnalysis metódust. A StopAnalyzingTweets metódus meghívja DestroyTimer metódust, beállítja az _isStopped adattagot igazra és naplózza, hogy az érzelemelemzési folyamat megállt. A DestroyTimer metódus, ha a _timer adattag nem null, akkor meghívja a rajta a Stop metódust.



29. Ábra TweetAnalyzerCoordinator activity diagram

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 adatbázisból 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 megállítjuk az időzítőt és az elemzési folyamatot. Ha nincs megállítás, akkor újból lekérdezzük a következő 50 darab tweetet.

A **TweetAnalyzer** osztály szerepe 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 érzellem 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 metódus paraméterként megkapja azt a tweetet amelynek a szövegén az érzelmi elemzést fogjuk végezni. Deklarálunk egy lokális változót, amely a talált érzelmek számát fogja nyilvántartani, `totalSentiments` néven kezdetben zérós értékkel. Egy `TextPreprocessor` típusú lokális változót is inicializálunk `preprocesser` néven. Lekérdezzük a `categoryIds` lokális változóba a tweetben szereplő összes kategóriát. A `categoryIds` lista minden elemére lekérdezzük az adott kategóriához tartozó kulcsszavakat, szinonimákat és tulajdonságokat, a `keywords`, `synonyms` és `characteristics` lokális listákba. A `keywords` és `characteristics` lista elemeiből elkészítjük a `keywordsForAnalyzer` és `characteristicsForAnalyzer` lokális listákat. A `preprocessor` lokális változón meghívjuk az `Preprocess` metódust átadva paraméterként a tweet szövegét, a szinonimák és a kulcsszavak listáját és visszakapunk egy `preprocessedText` lokális string típusú változót. Meghívjuk a `_textAnalyzer` adattag `Analyze` metódusát 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. Ha a `results` lista nem üres meghívjuk a `TrySaveEvaluatedResult` metódust. Frissítjük a `totalSentiments` változó értékét az újonnan sikeresen lementett érzelmi eredmények számával, majd visszatérítjük az értékét. A `TrySaveEvaluatedResult` metódus paraméterként megkapja az `EvaluationResult` típusú listát. Try-catch-ben naplózza az információt arról, hogy hány érzelmi elemzést fog lementeni az

adatbázisba és meghívja az `_evaluatedResultDao` adattagról a `SaveEvaluatedTweets` metódust továbbadva paraméterként az eredmények listáját. Hiba fellépése esetén naplózza az információt, hogy az adatbázisba nem sikerült lementeni az adatokat.

TweetAnalyzer
- <code>_logger: ILogger</code> - <code>_characteristicDao: CharacteristicDAO</code> - <code>_synonymDao: SynonymDAO</code> - <code>_keywordDao: KeywordDAO</code> - <code>_evaluatedResultDao: EvaluatedResultDAO</code> - <code>_textAnalyzer: TextAnalyzer</code>
+ <code>TweetAnalyzer(logger: ILogger): TweetAnalyzer</code> + <code>Analyze(tweet: Tweet): int</code> + <code>TrySaveEvaluatedResults(results: List<EvaluationResult>): void</code>

30. Ábra *TweetAnalyzer* osztály diagram

A **TextAnalyzer** osztály szerepe a tweet szövegének a Stanford Parser segítségével a mondattani és érzelmi kielemezése. 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` metódus paraméterként megkapja az elemzésre szánt szöveget, a kulcsszavak és tulajdonságok listáját. A `_relationFinder` adattagon meghívja a `FindRelations` metódust 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` metódus meghívásával, amelynek paraméterként átadjuk a szöveget és `relations` listát. A metódus eredményként visszatéríti a `result` nevű listát. A `CreatePipeline` metódus 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 modellek útvonalát.

TextAnalyzer
- <code>_pipeline: StanfordCoreNLP</code> - <code>_relationFinder: TextRelationFinder</code> - <code>_sentimentFinder: TextSentimentFinder</code>
+ <code>Analyze(text: string, keywords: List<Keyword>, characteristics: List<Characteristics>): IList<SentimentOfRelation></code> - <code>CreatePipeline(): StanfordCoreNLP</code>

31. Ábra *TextAnalyzer* osztály diagram

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` metódus paraméterként megkapja az elemzendő

szöveget, a kulcsszavak és a tulajdonságok listáját, végül visszatérít egy Relation típusú elemekből álló listát. Először meghívja a CoreMapForSentence metódust átadva paraméterként a szöveget és a visszatérített értéket tárolja a coreMap CoreMap típusú változóban. Ha ez a változó null, akkor a metódus visszatérít egy üres listát. Ha viszont van értéke, akkor meghívja a CollapsedCCProcessedDependenciesForCoreMap metódust átadva a coreMap változót, mint paraméter. A meghívott metódus eredményét a dependencies SemanticGraph típusú változóba tároljuk. A SemanticGraph típusú változót adjuk, mint paraméter a ConvertToAdjacencyGraph metódusnak a visszatérített értéket pedig a graph AdjacencyGraph típusú változóba tároljuk. Inicializáljuk a distancesByNoun Dictionary<WordVertex, List<Distance>> típusú változót. Meghívjuk a FindNounVertices metódust, átadva a graph változót paraméterként, a visszakapott WordVertex típusú elemekből 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. Létrehozunk 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 metódus paraméterként megkapja az elemezendő szöveget és visszatéríti a Stanford Parser által annotált mondatot, az annotációk típusát a _pipeline adattag tartalmazza. A CollapsedCCProcessedDependenciesForCoreMap metódus paraméterként megkapja a coreMap változót és ez alapján ugyancsak a Stanford Parser-t használva visszatérít egy SemanticGraphCoreAnnotations.CollapsedCCProcessedDependenciesAnnotation típusú szemantikus gráfot, abban az esetben, ha a coreMap paraméternek van értéke. A ConvertToAdjacencyGraph metódus megkap egy SemanticGraph típusú paramétert és ezt átalakítja AdjacencyGraph<WordVertex, SEdge<WordVertex>> típusú gráffá. A WordVertexFromIndexedWord metódus az IndexedWord típusú paramétert alakítja át saját WordVertex típusú csúcsnak. A FindNounVertices metódus a AdjacencyGraph típusú

paraméterből előkészít egy WordVertex típusú elemekből álló listát ami tartalmazza a gráfban szereplő összes főnevet. A CalculateDistancesToNouns metódusnak átadjuk paraméterként egy AdjacencyGraph típusú gráfot és egy WordVertex típusú csúcsot. A metódus felhasználva a QuickGraph-nak a ShortestPathsDijkstra metódusát felépít egy Distance típusú elemekből álló listát, ami tartalmazza a mondatban szereplő összes főnevek közti legkisebb távolságokat.

TextRelationFinder
- _pipeline: StanfordCoreNLP
+FindRelations(text: string, keywords: List<Keyword>, characteristics: List<Characteristic>): IList<Relation>
-CoreMapForSentence(text: string): CoreMap
-CollapsedCCProcessedDependenciesForCoreMap(coreMap: CoreMap): SemanticGraph
-ConvertToAdjacencyGraph(graph: SemanticGraph): AdjacencyGraph<WordVertex, SEdge<WordVertex>>
-WordVertexFromIndexedWord(word: IndexedWord): WordVertex
-FindNounVertices(graph: AdjacencyGraph<WordVertex, SEdge<WordVertex>>): IList<WordVertex>
-CalculateDistancesToNouns(graph: AdjacencyGraph<WordVertex, SEdge<WordVertex>>, source: WordVertex): IList<Distance>

32. Ábra TextRelationFinder osztály diagram

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 metódus paraméterként megkapja az elemzésre szánt szöveget és egy Relation típusú elemekből 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 metódus paraméterként megkapja az elemzendő szöveget és visszatéríti a Stanford Parser által annotált mondatot, az annotációk típusát a _pipeline adattag tartalmazza.

A FindRelationWordsAndVerb metódus 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 szöveggént tartalmaznak egy-egy mondatrészt, amint ezt a 3.5 fejezetben már ismertettem. A metódus addig halad a fa csomópontjain, míg egy

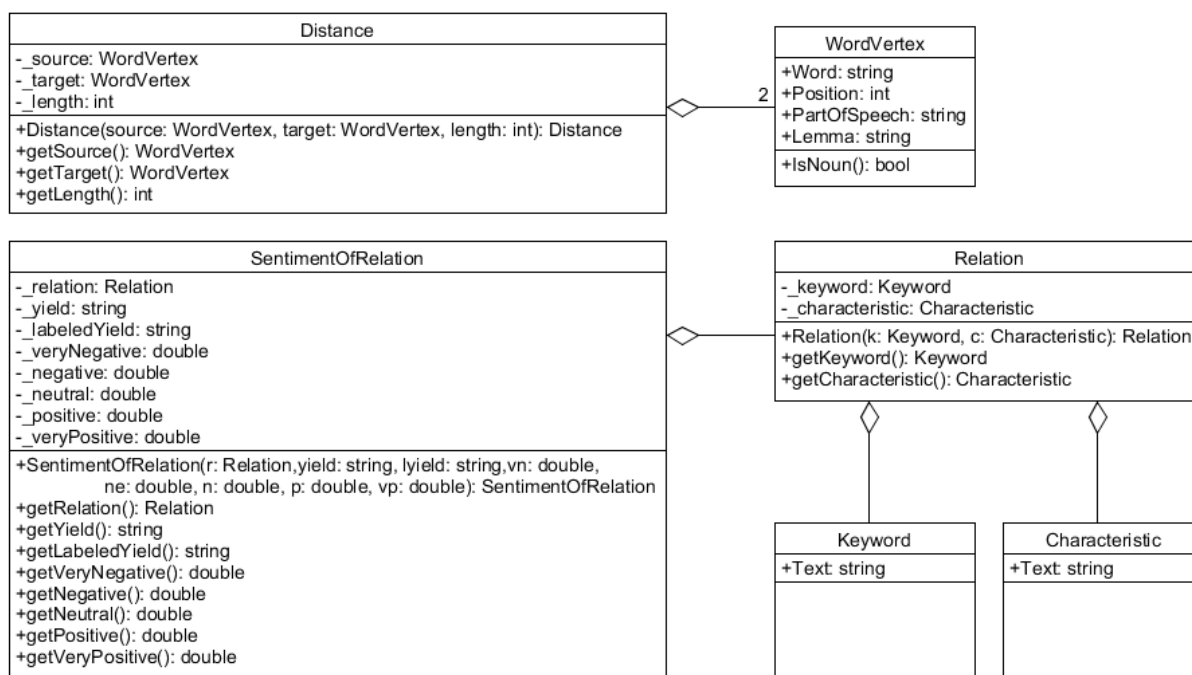
olyan csomópontot nem talál, melyben szerepel a kulcsszó, a tulajdonság és egy ige. Ennek a csomópontnak az érzelmi értékét téríti vissza eredményként.

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

TextSentimentFinder
- _pipeline: StanfordCoreNLP
+TextSentimentFinder(pipeline: StanfordCoreNLP): TextSentimentFinder
+FindSentiments(text: string, relations: IList<Relation>): IList<SentimentOfRelation>
-CoreMapForSentence(sentence: string): CoreMap
-FindRelationWordsAndVerb(root: LabeledScoredTreeNode, tokenAnnotations: List<CoreLabel>, relation: Relation)
-IsVerb(ner: string)

33. Ábra TextSentimentFinder osztály diagram

A segéd modell osztályok azért voltak létrehozva, hogy az érzelelemző 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.



34. Ábra Segéd modell osztályok

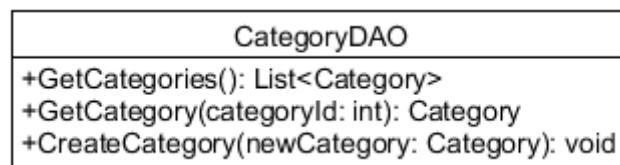
A Keyword és Characteristic osztályok egy-egy karakterláncot tárolnak. A Relation osztály egy kulcsszó és karakterisztika közti relációt modellez. A SentimentOfRelation osztály egy relációt tartalmaz, 2 mondatrészletet és az érzelmi értékeket százalékosan kifejezve. A yield mondatrészlet, a mondat azon 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”.

4.2.4 Data access komponens

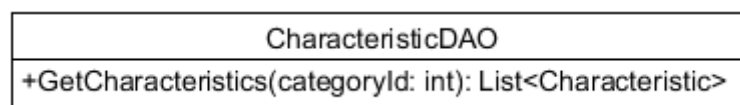
A Data access komponens szerepe az applikáció és adatbázis közti kommunikáció megvalósítása. Ebben a komponensben lévő osztályok segítségével az adatbázis tábláira végzünk lekérdezéseket, viszünk be új adatokat, módosítjuk a létező adatokat és törölünk az adatokból.

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.



35. Ábra CategoryDAO osztály diagram

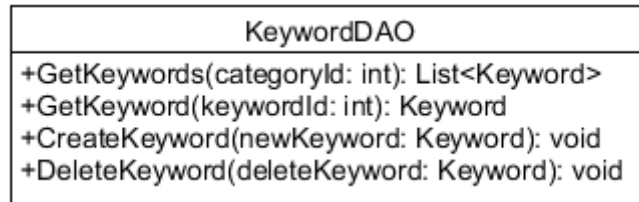
A CharacteristicDAO osztály az adatbázisban a Characteristic táblához biztosít hozzáférést. A GetCharacteristics metódus visszatéríti egy listában mely elemei Characteristic típusúak a paraméterként megkapott kategóriának megfelelő tulajdonságokat.



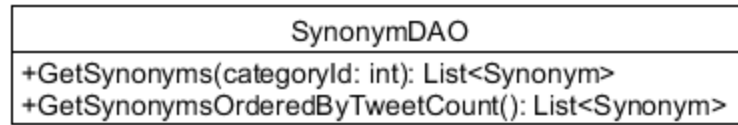
36. Ábra CharacteristicDAO osztály diagram

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 törli 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.

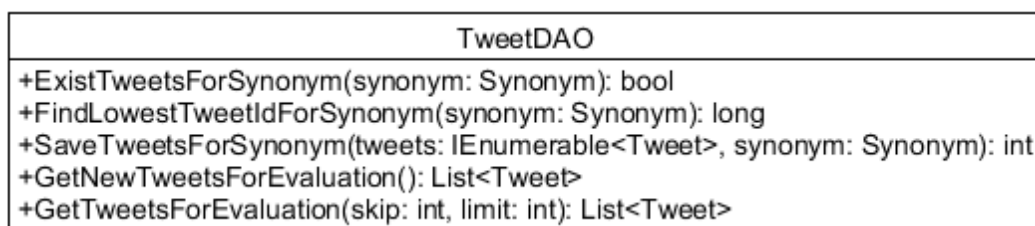


37. Ábra KeywordDAO osztály diagram



38. Ábra SynonymDAO osztály diagram

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 érzelemelemzést. A GetTweetsForEvaluation metódus visszatérít egy Tweet típusú elemeket tartalmazó listát, amelyben az ID szerint rendezett tweetekből megmondjuk, hogy hány tweetet szökünk át és leg több hány darab tweetet kérünk a listában. Erre szolgálnak a skip és limit paraméterek.



39. Ábra TweetDAO osztály diagram

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 weboldalokról származó adatok szövegeire voltak végrehajtva. A SaveEvaluatedResult metódus beszúrja a táblába a paraméterként kapott EvaluationResult típusú elemeket tartalmazó lista elemeit.

EvaluatedResultDAO
+SaveEvaluatedTweets(results: IEnumerable<EvaluationResult>): void +SaveEvaluatedWebData(results: IEnumerable<EvaluationResult>): void -SaveEvaluatedResult(webDataType: int, results: IEnumerable<EvaluationResult>): void

40. Ábra EvaluatedResultDAO osztály diagram

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

WinServicesDAO
+IsTweetDownloaderServiceEnabled(): bool +IsTweetAnalyzerServiceEnabled(): bool +EnableTweetDownloaderService(): void +DisableTweetDownloaderService(): void +EnableTweetAnalyzerService(): void +DisableTweetAnalyzerService(): void -IsServiceEnabled(serviceld: int): bool -EnableService(serviceld: int): void -DisableService(serviceld: int): void

41. Ábra WinServicesDAO osztály diagram

4.2.5 A model komponens

A model komponens tartalmazza az adatbázisban szereplő táblák és tárolt eljárások osztályokká való leképezését. Az adatbázisban szereplő táblák részletes leírása megtalálható a 4.2.2 fejezetben, a 10 táblázatban, valamint az egyed-kapcsolat diagram a 16 ábrán.

4.3 A rendszer felhasználása

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

-
- adminisztrációs
 - szervizek üzemeltetése
 - eredmények vizsgálata

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

A szervizek üzemeltetésének a beállításával a felhasználó elindíthatja vagy megállíthatja a két szervizt, a tweetek gyűjtését és az érzelelemzést. A tweet gyűjtési szerviz engedélyezésével elindul a tweetek keresése és letöltése az internetről. Az applikáció a felhasználó által definiált szinonimákat fogja keresni a tweetek szövegében és a találatokat lementi egy saját adatbázisba. Az érzelem elemzési szerviz engedélyezésével elindul a letöltött és még ki nem elemzett tweetek szövegének az érzelmi kiértékelése.

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.

4.4 Üzembe helyezés és kísérleti eredmények

4.4.1 Felhasznált technológiák

A következőkben a rendszerben használt technológiákat mutatom be. A rendszer web alkalmazás komponense a Microsoft által létrehozott ASP.NET MVC 4-es verziójú technológiára épül. A web alkalmazás 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 valósítja meg. A rendszer két fontos komponense, az adat letöltés és az elemzés, ami Windows szervizként 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 elemző komponens a 3.5.1-es verziójú Stanford Core NLP technológiára épül. Ugyanez 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.

4.4.2 Felmerült problémák és megoldásai

A következőkben leírom, hogy a rendszer megvalósítása során milyen 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, hashtagek. Ezekre a szavakra nem igazán végzett helyes szófaji annotációt a Stanford Parser szövegelemzője. A probléma megoldására egy szöveg elő feldolgozást használtam, amely során a helyettesítem a http-vel kezdő linkeket a LINK, a @ jellel kezdődő szövegrészeket a NAME, valamint a # jellel kezdődő szövegrészeket a CATEGORY szavakkal.

A negyedik problémát a szervizek webes felületről történő elindítása jelentette. Mivel a szervizek Windows szervizként 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 applikáció, mint a windows szervizek hozzá kell férjenek. Ezt a flag-et a web applikáción keresztül módosíthatja a felhasználó. A szerviz egy előre definiált időközönként ellenőrzi az adatbázisban a flag értékét és ennek megfelelően elindítja a munkáját vagy megállítja azt.

4.4.3 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 tartalmazznak 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):

11. Táblázat Relációkeresés teszt eredmények

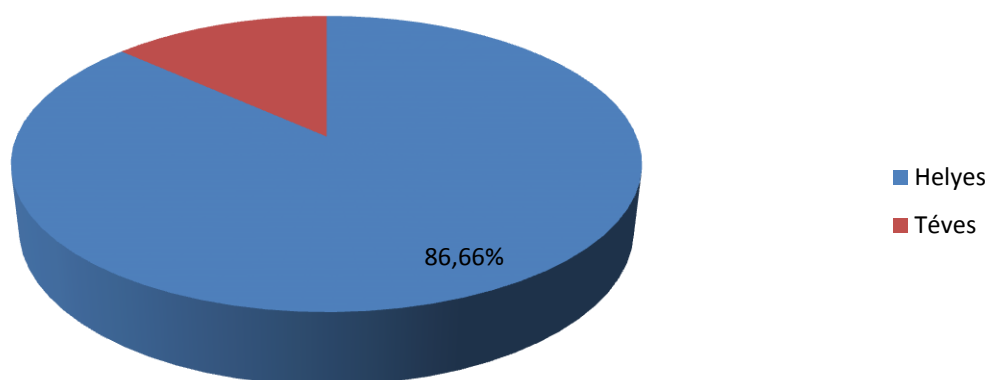
IK	IT	VK	VT	Mondat
2	2	2	2	The engine of Audi is great, but VW has the worst brakes.
2	2	1	2	The brake and engine of this Audi is great.
2	2	2	1	The brake of this Audi and VW is great.
2	2	2	2	Audi and VW have good engines and brakes.
2	2	2	1	Audi has better brakes than VW.
2	2	2	2	Audi has a powerful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



42. Ábra Teszteredmények sikeressége

A relációkereső olyan esetekben téved, amikor a mondatban 2 kulcsszó és 2 tulajdonság szerepel, és a kereső ismeri az egyik kulcsszót 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.

5 Következtetések

A dolgozatban bemutatott rendszer lehetőséget ad a felhasználónak saját keresési kulcsszavakat tartalmazó tweetek keresésére és letöltésére a Twitter-ről. Továbbá a felhasználó saját tulajdonság - listát készíthet, ami az érzelemelemzés célját fogja alkotni. A rendszer a tweetek szövegét szófaji és relációfüggőségek elemzésével feldolgozza, kikeresi azokat a szövegrészeket amelyek, tartalmaznak kulcsszó-tulajdonság relációt. Az ily módon előkészített szövegre érzelem kiértékelést végez. A felhasználó az elvégzett kiértékelések eredményeit összehasonlító táblázatként elemezheti.

A rendszer hasznossága abban nyilvánul meg, hogy lehetőséget ad két vagy több kulcsszó meghatározásával (például Audi és Volkswagen), valamint tulajdonságok meghatározásával (például motor, biztonság, végsebesség), összegyűjthetjük a kulcsszavakat tartalmazó Tweeteket. Az érzelelemzés után pedig hasznos információkat kapunk arról, hogy a felhasználók mit is gondolnak, az adott tulajdonságról. Láthatjuk, hogy az autómárkák közül ki milyen tulajdonságra milyen eredményt ért el, miben jó, miben gyengébb. Ez az információ hasznos a vásárlók szempontjából hisz megtudhassák melyik autómárkák értek el jobb eredményt és melyik tulajdonságokra. Ez alapozhassa meg a döntésüket autóvásárlás esetén. Az autógyárak szempontjából is hasznos ez az információ, hisz ezek alapján hozhatnak döntést arról, hogy mit kell feljavítani, mivel elégedetlenek a felhasználók, miben ért el a konkurencia jobb eredményt. A rendszer elkészítése során megtanultam, hogy mennyire összetett, egy érzelelemző rendszer felépítése. Elsősorban szükség van egy adatbányászati modulra, melynek segítségével összegyűjtjük az adatokat. Ezeket egy szövegelemző segítségével feldolgozzuk, majd érzelmileg ki kell értékelnünk. A Stanford szövegelemzője által megtanultam, hogyan is működik a számítógépes szövegelemzés, mely elsősorban a mesterséges intelligenciára épül. A szövegelemzés eredményeként egy olyan fát kapunk, mely a mondatokban szereplő szavak közti relációkat tükrözi. Az érzelmi kiértékelés, szintén a mesterséges intelligencián alapszik, előre definiált modelleken, és a neuronhálókhoz hasonlóan tanítva van. Végül azt tanultam még, hogy a mondatban lévő szavak keresésekor, előnyös a szavakat a lemmatizált alakra hozni és úgy végezni a keresést, mivel így pontosabb a találat, mint a hagyomány karakterláncbeli kereséssel.

5.1 Megvalósítások

A rendszer egy több komponensből álló applikáció, melynek a felhasználói felülete egy web alkalmazás, ami C# nyelvben íródott 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 perzisztálja.

5.2 Hasonló rendszerekkel való összehasonlítás

Sentimentor – egy olyan eszköz, amely a Twitter adatok érzélem elemzését végzi. [19] Ez az eszköz a naiv Bayes Classifier-t használja arra, hogy a tweeteket pozitív, negatív és objektív osztályokba 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-applikáció, amely a szavak előfordulásáról készít kimutatásokat, érzelelemelmezést végez szöveg-részre, keresést hajt végre a Tweetek között.

Az Umigon applikáció szókincs és heurisztikára alapozott érzelelemelmezést végez a tweeteken.

[20] Speciálisan arra volt tervezve, hogy a tweetekben a pozitív, negatív vagy semleges érzelmekeket 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 előző lépések során felfedezett szemantikus jellemvonásokat, elvégzi a végleges kiértékelést a teljes tweetre és egyetlen érzelmet csatol a tweetnek

Az általam készített applikáció bizonyos szempontokból hasonlít a megemlített applikációkra és vannak olyan részek, amelyekben különbözik.

Hasonló részek:

- A szöveg előkészítése érzelelemelmezésre
- 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 érzelelemelmező fő sajátossága abban van, hogy nem a teljes mondatra végez érzelem kiértékelést, hanem a kulcsszó – tulajdonság reláció megkeresése után előállított szövegrészre.

Amint azt már az előző fejezetekben is kihangsúlyoztam, ahhoz, hogy az érzelelemelmezés minél pontosabb legyen, fontos, hogy ezt csak a mondat bizonyos részeire végezzük el. Éppen e miatt, a rendszerem tervezésekor, bevezettem a névelem és tulajdonság párosokat, melyekkel a felhasználó definiálhatja egy adott témakörön belül mikről fogalmazott véleményekről és egész pontosan azoknak mely tulajdonságaikról érdekelt. Ezzel a rendszerrel érzelelemelmezést végezhetünk az autógyártók autóinak különböző alkatrészeiről és tulajdonságairól, mint pl. fék, motor, irányíthatóság, biztonság stb. Ezt az előbb említett rendszerek nem teszik lehetővé. Úgy gondolom, hogy ez egy olyan újítás, mely által a felhasználó sokkal pontosabb információt kap az őt érdekelt termékekről, lebontva akár ezek tulajdonságaira is.

Fontosnak tartom azt is megemlíteni, hogy a rendszernek korlátai is vannak, melyek közül az első a tweet-ek egyszeri kiértékelése. Úgy gondolom, hogy fontos lenne a tweet-eket újraértékelni, mivel a rendszerben a kulcsszavak és tulajdonságok listája folyamatosan bővül. A 4.4.3 fejezetben már ismertettem, hogy a relációkeresési algoritmus többnyire akkor téved, amikor túl kevés az általa ismert kulcsszó és tulajdonság, ezért ha a rendszer bővül, egyre pontosabb lenne a relációkeresési algoritmus. A második korlátja a rendszernek az, hogy csak visszamenőleg gyűjt tweet-eket a keresés által, mely a Twitter API leírása szerint, nem az összes létező tweet között keres. A harmadik korlát, hogy a rendszerbe kerülő adatoknak csak egyetlen forrása van, a Twitter. A rendszer felhasználhatna több forrást is, mint például egy adott témakörre szakosodott fórumok bejegyzéseit, vagy más szociális hálókat.

5.3 További fejlesztési irányok

A következő alfejezetekben, leírom, hogy a rendszer mely részeit és milyen irányba fejleszthetném tovább.

5.3.1 Tweetek gyűjtése Stream API-val

Az applikáció tweet gyűjtő moduljának a bővítése a Tweet API által biztosított Streaming API használatával. Ily módon 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.

5.3.2 Tweet szöveg újra - kiértékelése

Egy másik tovább fejlesztési lehetőség az lenne, hogy azokat a tweeteket amiket már kiértékelünk 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 előzőleg kiértékelte 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.

5.3.3 Adatforrások bővítése

Az applikáció csak a Twitteren megjelent tweeteket használja adatforrásául, de szerintem sokkal bővebb és talán szakszerűbb 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 beleegyezése.

5.3.4 Rendszerállapot kijelzés

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

6 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] E. Cambria, B. Schuller, Y. Xia és C. Havasi, „New Avenues in Opinion Mining and Sentiment Analysis,” *IEEE Computer Society*, 2013.
- [6] M. A. Russell, *Mining the Social Web*, Sebastopol: O'Reilly Media, 2011.
- [7] X. Ding és B. Liu, „The Utility of Linguistic Rules in Opinion Mining,” *Department of Computer Science, University of Illinois at Chicago*, 2007.
- [8] M. Z. Arbi, „Natural Language Processing,” *Mes recherches personnelles*, 2015.
- [9] S. Monika, *Bevezetés a korpusznyelvészeti*, Budapest: Tinta Könyvkiadó, 2005.
- [10] Stanford Core NLP Függőségi Tankönyv, 2015.
- [11] B. Liu, *Sentiment Analysis and Opinion Mining*, Morgan & Claypool Publishers, 2012.
- [12] 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*.
- [13] M.-C. de Marneffe és D. C. Manning, „The Stanford typed dependencies representation,” *Proceeding CrossParser '08 Coling*, 2008.
- [14] „Stanford Sentiment Analysis honlap,” [Online]. Available: <http://nlp.stanford.edu/sentiment/index.html>.
- [15] 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*.
- [16] A. Agarwal, B. Xie, I. Vovsha, O. Rambow és R. Passonneau, „Sentiment Analysis of Twitter Data,” *Department of Computer Science, Columbia University*, 2011.

-
- [17] A. Pak és P. Paroubek, „Twitter as a Corpus for Sentiment Analysis and Opinion Mining,” *Universite de Paris-Sud, Laboratoire LIMSI-CNRS*, 2010.
- [18] 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.
- [19] J. Spencer és G. Uchyigit, „Sentimentor: Sentiment Analysis of Twitter Data,” *School of Computing, Engineering and Mathematics, University of Brighton*, 2012.
- [20] 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.