UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Colectarea si analizarea datelor de pe reddit

propusă de

Cristian-Andrei Ursu

Sesiunea: *iulie, 2019*

Coordonator științific

Conf. Dr. Anca Vitcu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Colectarea si analizarea datelor de pe reddit

Cristian-Andrei Ursu

Sesiunea: *iulie, 2019*

Coordonator științific

Conf. Dr. Anca Vitcu

Avizat,

Îndrumător Lucrare de Licenţă

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

DECLARAŢIE privind originalitatea conţinutului lucrării de licenţă

Subsemntatul(a) ……

domiciliul în ……..

născut(ă) la data de ……., identificat prin CNP ……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …… specializarea ……., promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*Titlul complet al lucrării*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iaşi, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *data*

Absolvent *Prenume Nume*

_____

(semnătura în original)

Contents

Introducere

RCruncher este o aplicatie web ce se ocupa cu colectarea si analiza datelor de pe reddit. Functionalitatea aplicatiei este impartita in doua arii: colectarea si analiza datelor pentru utilizatorii de reddit si colectarea si analiza datelor pentru asa numitele subreddits.

In ceea ce priveste utilizatorii, functionalitatea consta in a le oferi subreddits in care acestia ar putea fi interesati bazat pe activitatea anterioara in cadrul siteului si de a le indica utilizatorii cu interese asemanatoare. In ceea ce priveste postarile propriu-zise de pe reddit, postarea este analizata din punct de vedere semantic si din punct de vedere a analizei sentimentelor.

Din punct de vedere al originalitatii, reddit implementeaza unele dintre functionalitatile pe care le propune RCruncher dar nu ofera niciun fel de aplicatie - externa spre acest scop. Ca aplicatii externe oferite de redidt, ce nu folosesc protocolul OAuth, majoritatea sunt wrappers peste API-ul expus de reddit.

Pentru a putea furniza solutii la problema propusa, aceea de a oferi recomandari si de a analiza anumite posture, am construit o aplicatie web ce consta in server-client ce serveste acest scop. Pentru inceput, se foloseste API-ul expus de catre reddit pentru a colecta datele ce tin de utilizatori, spre exemplu: posturile create si in ce subreddits si comentariile lasate si la ce posturi. Aceste date sunt mai tarziu salvate intr-o baza de date. Pe datelor colectate, cu ajutorul unei retele neuronale, se ofera predictiile, recomandariile si utilizatorii inruditi. Pentru cea de a doua parte, se foloseste un crawler si API-ul extern pentru a obtine continutul propriu zis al postarilor dupa care datele sunt analizate de catre API-ul de limbaj natural al celor de la google, datele fiind salvate anterior in baza de date anterior mentionate.

Ca si capitole ale acestei lucrari vor fi abordare urmatoarele:

- Serverul si arhitectura acestuia.
- Construirea si antrenarea retele neuronale.

- Exemplificarea folosirii API-ului celor de la Reddit si a crawler-ului.
- Exemplificarea si argumentarea folosirii API-ului de la google.
- Baza de date.
- Conexiunea cu baza de date.
- Clientul si arhitectura acestuia.
- Lipsuri si posibile imbunatatirii ale aplicatiei.

Contributia proprie in dezvoltarea aplicatiei

In dezvoltarea aplicatiei, asupra partii de server a fost concentrate mare parte din efort. Astfel in ceea ce priveste serverul, singura dependenta este NestJs, tot ce tine de designul architectural al serverului, controlerele si metodele respective acestora, domeniul si arhitectura lui, serviciile folosite, toate sunt aproape in totalitate munca proprie.

In ceea ce priveste reteaua neuronala, alegerea unei retele potrivite pentru setul de date disponibil, pregatirea mediului favorabil, alegerea si pregatirea datelor pentru antrenare, alegerea optiunilor pentru creare, documentarea asupra functiei de distanta si alegerea acesteia, integrarea si adaptarea retelei sunt munca proprie, depinzand totusi de un pachet extern ce reprezinta o baza a retelei neuronale de tip Kohonen.

Designul bazei de date, crearea tabelelor, a dependentei dintre acestea si a operatiilor efectuate asupra acesotra sunt dependente de tooluri externe, insa contributia proprie este semnificativa.

In ceea ce priveste colectarea datelor si analiza cu instrumentele de la google: instantierea apelurilor, validarea si prelucrarea datelor atat primite cat si trimise, reprezinta principala contributie in acest domeniu.

Pe partea de client, pentru partea de vizualizare sunt folosite unele librarii externe insa pentru care este necesara partea de conexiune cu serverul ce consta in serviciile de la nivelul clientului si prelucrarea si validarea datelor.

Serverul si arhitectura acestuia

Partea de server este construita dupa principiul dupa DDD(domain-driven design), astfel incat concentrearea este directionata catre domeniu si logica acestuia.Pe partea de server am optat sa folosesc node.js si typescript in detrimentul PHP-ului.

Printre motive se enumera:

- Folosirea aceluiasi limbaj de programare la niveul clientului si al serverului.
- Moulele incarcate de catre node sunt descarcate si dupa initializate, ulterior fiind disponibile constant.
- Se lucreaza mai usor cu fisiere de dimensiuni mari.
- Caracterul strongly-typed al typescriptului fata de javascript
- Numeroasele module oferite de catre Node.js
- Debugging in real time

Serverul este construit cu ajutorul lui NestJS, un framework specializat pe constructia de servere, care poate fi descris cel mai bine ca un wrapper peste Express. Hostarea acestuia are loc pe portul 3000 si reprezinta punctul de start al aplicatiei.

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

Aplicatia contine un singur modul in care se intializeaza dependentele acesteia, de exemplu: modulul de CQRS, conexiunea cu baza de date, controllerele si serviciile de care depinde modulul.

```
@Module({
  imports: [TypeOrmModule.forRoot(), CqrsModule],
  controllers: [RedditUsersController, RedditPostsController],
  providers: [
    ...CommandHandlers,
    ...QuerryHandlers,
    RedditDataService,
    TextEnchancerService,
    NaturalLanguageService,
    CrawlerService,
    PageContentRequester,
  ],
})
export class AppModule {
  constructor(private readonly connection: Connection) { }
}
```

API-ul expus de catre server este unul de tip REST. Acesta expune doua controlere:

- Pentru userii de reddit

```
@Controller('reddit-users')
```

- Pentru posturile de pe reddit

```
@Controller('reddit-posts')
```

Majoritatea metodelor expuse definite in controller-ul pentru utilizatori iar amandoua controller-ele contin metode doar de tipul POST si GET.

Pentru controller-ul ce corespunde utilizatorilor urmatoarele metode sunt expuse:

- Create, de tipul POST care primeste numele unui utilizator de reddit, creeaza entitatea corespunzatoare si o salveaza in baza de date.
- GetUserData, de tipul GET, care returneaza din baza de date un utilizator si toate datele legate de acesta.

- PredictUser, de tipul GET, care indica pozitia utilizatorului in cadul retelei neuronale.
- GetUserTopics, de tipul GET, care returneaza topicurile unui user bazate pe comentariile lasate de acestea in scopul construirii unui nor de cuvinte.
- GetTrainingSet, de tipul GET, care returneaza pozitiile tuturor utilizatorilor in cadrul retelei neuronale, utilizatori ce au facut parte din setul de initial de antrenament.
- RefreshComments, RefreshSubmitted, RefreshTopics, de tipul GET, care ii comunica serverului ca pentru un anumit utilizator se doreste innoirea respectivelor date.

Pentru controller-ul ce corespunde postarilor urmatoarele metode sunt expuse:

- CreatePost, de tipul POST, ce primeste URL unui post de pe reddit, il trimite catre analiza si ulterior catre salvarea lui in baza de date impreuna cu analiza corespunzatoare.
- GetPostWithData, de tipul GET, ce primeste URL unui post de pe reddit si returneaza date despre acesta impreuna cu analiza corespunzatoare a postarii.

Fiecare serviciu de la nivelul serverului, mai putin reteaua neuronala, este de tip Injectable, ceea ce inseamna ca, similar unui singleton, fiecare serviciu Injectabil v-a exista intr-o singura instantiere pe parcursul unei sesiuni de viata a serverului.

Toate functiile asincrone de pe server, cum ar fi aducerea datelor de pe reddit sau returnarea unor date la nivelul API-ului, folosesc tipul de date Observable din libraria Rx.js, ce reprezinta o inbunatatire de la Promise, sau patternul async, await ce sunt deseori folosite in aplicatii bazate pe node.js atunci cand este nevoie de tratarea asincronismului.

Pentru operatiilor asupra bazei de date se foloseste sablonul CQRS, care vizeaza segregarea comenzilor de interogari. In acest scop, pentru fiecare interogare sau comanda exista cate un handler si un query sau respectiv command. Astfel clasa command sau query create contine datele necesare procesarii, iar in clasa handler are loc procesearea propriu zisa a comenzilor. De exemplu, pentru a colecta comentariile unui utilizator de reddit in baza de date, au fost create urmatoarele clase:

- AddCommentsForFirstTimeRedditUserCommand
- AddCommentsForFirstTimeRedditUserHandler

Astfel, comanda contine numele utilizatorului reddit pentru care se colecteaza comentariile, care este unic:

```
export class AddCommentsForFirstTimeRedditUserCommand {
  constructor(public readonly redditUser: RedditUserModel) { }
}
```

Clasei din urma ii revine responsabilitatea de a colecta datele, de ale prelucra, si de ale salva in baza de date. Pentru interogari structura este asemanatoare.

Pentru a executa aceste comenzi si interogari, sunt folosite urmatoarele doua servicii injectabile: CommandBus si QueryBus, folosite in ambele controlere.

Chiar daca, un asemenea bus pare sa aiba la inceput doar responsabilitatea de a lega comenzile si interogarile de handlere, utilitatea acestuia este mult mai larga. Acesta valideaza datele din interiorul unei comenzi, incapsuleaza handlerul intr-o tranzactie a bazei de date si pastreaza ordinea comenzilor/interogarilor, respectand astfel sablonul lantului de responsabilitate.

Construirea si antrenarea retelei neuronale.

In ceea ce priveste reteaua neuronala am optat pentru una de tip SOM(self-organizing map)/ Kohonen din mai multe motive:

- Reprezinta un tip de retea neuronala cu invatare nesupervizata.
- Este disponibila alegerea unei functii de distanta customizata.
- Tipul de invatare oferit de retea este : invatare competitive in opozitie cu invatarea bazate pe corectare de erori. Am optat pentru aceasta varianta deoarece cu greu se poate vorbi de erori in cazul de recomandari sau predictii.
- Este indicata in cazul vrem sa vizualizam predictii pentru date foarte mari. Pentru o retea neuronala construita din 100 de utilizatori si 633 de sub topics reteaua neuronala, salvata sub forma unui fisier de tipul .JSON ajunge la o marime de aprox. 85 MB, in conditiile in care, in martie 2019, reddit a inregistrat 542 de milioane de utilizatorii.

Chiar daca este privita ca un serviciu, am ales sa nu ii ofer retelei neuronale caracterul de clasa

Injectabila, datorita timpului lung de incarcare si antrenare a acesteia, ci mai degraba sa o instantiez la crearea controllerului pentru utilizatori de care este strans dependenta. Astfel reteaua va fi incarcata la pornirea serverului si nu cand v-a fi nevoie de ea propriu-zis.

Setarile dupa care este construita reteaua sunt urmatoarele, urmand sa fie explicate in urmatoarele cateva paragrafe :

```
export class KohonenOptions {
    public fields;
    public iterations = 100;
    public learningRate = 0.1;
    public xValue = 100;
    public yValue = 100;
```

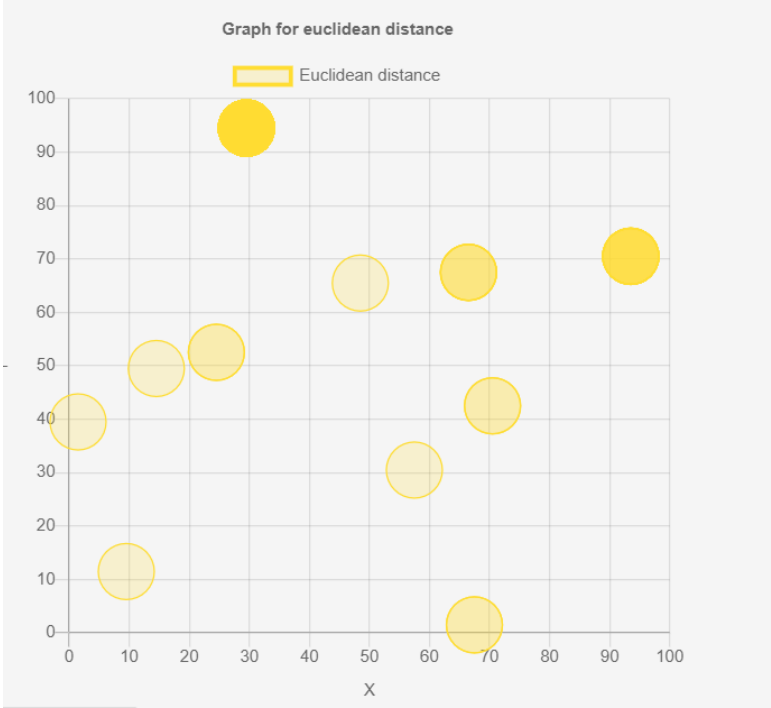
Astfel, reteaua neuronala consta in 100*100 de celule. Iteratiile sunt de asemenea 100, ceea ce inseamna ca se va itera de 100 * Lungimea datelor de antrenament in faza de antrenare a retelei. Coeficentul de invatare este 0.1, acesta fiind transmis mai departe catre algoritmul ce se ocupa cu invatarea.

Astfel, pentru a putea antrena reteaua sau pentru a putea produce o recomandare/prezicere este nevoie de asesarea datelor in urmatorul format: pentru fiecare utilizator este creat un json cu urmatorul tip de inregistrari : numeSubreddit : [0, PMAX], unde PMAX reprezinta numarul maxim de postari facute de orice utilizator in acel subreddit. Astfel pentru fiecare utilizator for fi parcuse toate subredditurile si create inregistrarile necesare. Daca un utilizator nu a postat niciodata intr-un subreddit, activitatea acestuia in subredditul corespunzator este considerate nula.

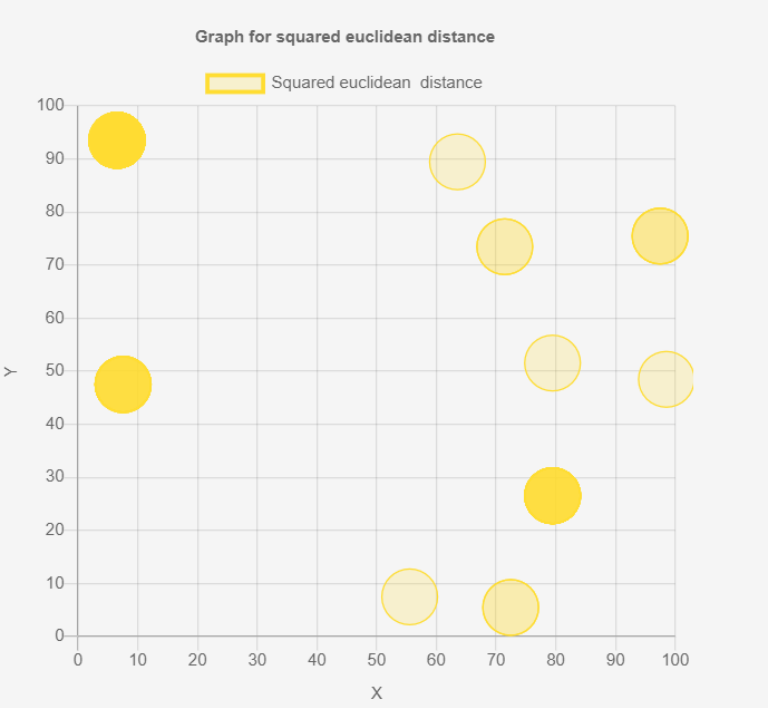
```
public buildUserTrainingSet(currentUser: RedditUserEntity, allSubbreddits: UserSubredditEntity[]): Observable<any> {
    const newPromise = new Promise(async (resolve) => {
        const userDataSet = {};
        for (const subreddit of allSubbreddits) {
            const foundSubredditForUser = await UserSubredditEntity.findOne(
                {
                    where: { origin: subreddit.origin, owner: currentUser },
                },
            );
            if (this.baseFields.includes(subreddit.origin)) {
                if (foundSubredditForUser !== undefined) {
                    userDataSet[subreddit.origin] = subreddit.numberOfAppearances;
                } else {
                    userDataSet[subreddit.origin] = 0;
                }
            }

        }
        resolve(userDataSet);
    });
    return from(newPromise);
}
```

In continuare vom aborda functia de distanta dintre utilizatori si procesul alegerii acesteia.

In urmatoarele grafice este evidentiata distrbutia a 100 de utilizatori cu 633 de subreddituri, singura schimbare fiind functia de distanta.



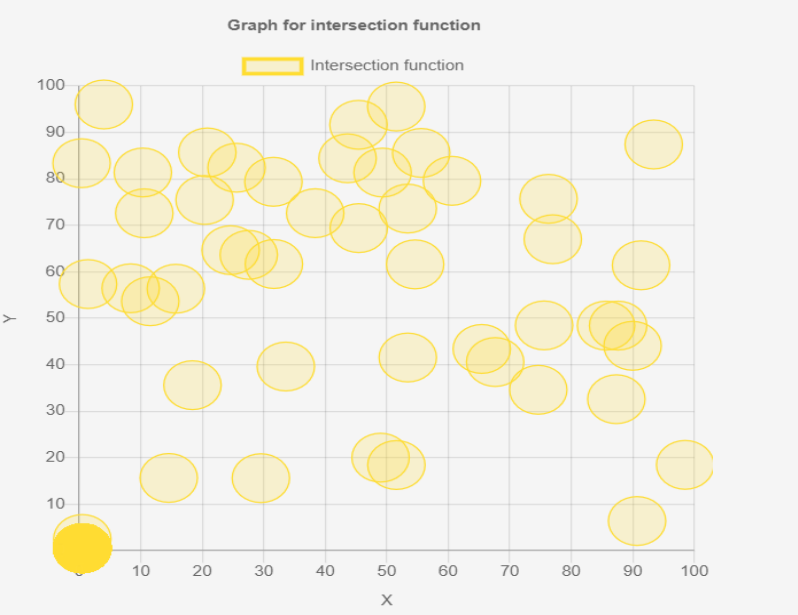
In cazul distantei euclidiene putem observa o dispersie inadecvata datelor, formandu-se doar doua clustere fixe, unul dintre ele fiind pentru utilizatorii pentru care nu se inregistreaza niciun fel de postare in cadrul redditului.



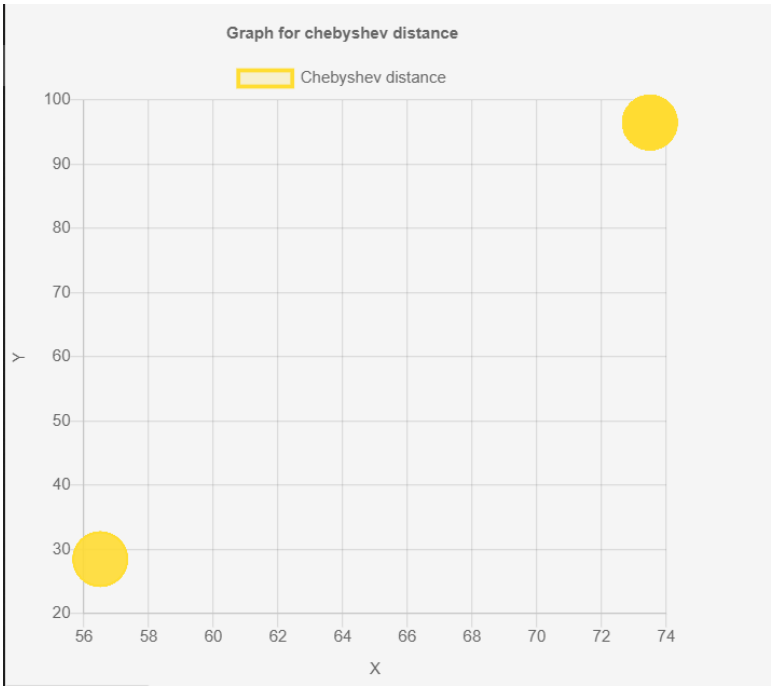
In cazul radacinii patrate a distantei euclidiene, se poate constata aparitia unui al 3 cluster, tot fix, dispersia datelor in grafic fiind inca inadecvata.

In ciuda faptului ca radacina patrata a distantei euclidiene reprezinta distanta de baza ce se foloseste in cadrul multor algoritmi de invatare automata, aici se poate observa cu usurinta gradul ridicat de inadecvare a acesteia din constructia datelor pentru fiecare utilizator.

In continuare am apelat tot la o functie de baza, si anume, intersectia intre doua inregistrari.

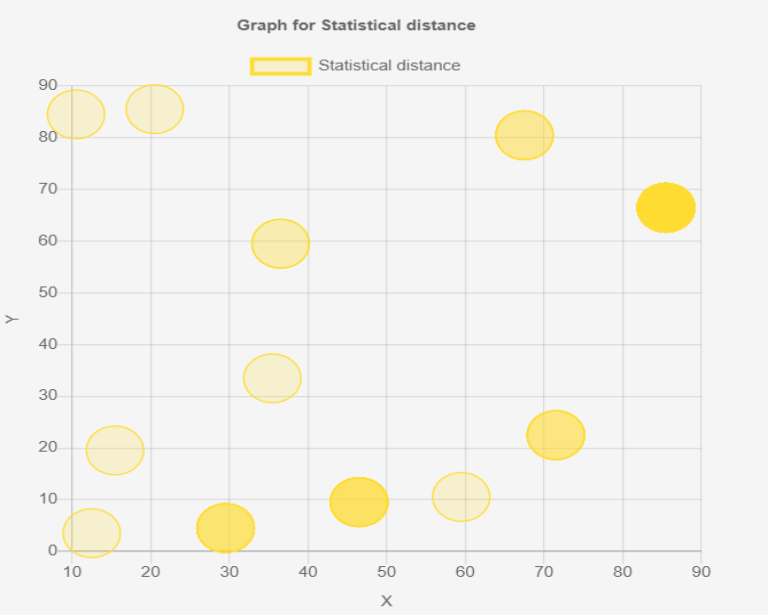


In cazul intersectiei se observa o dispersie mai buna a datelor, un singur cluster fix, si altele 2-3 clustere mai dispersate.

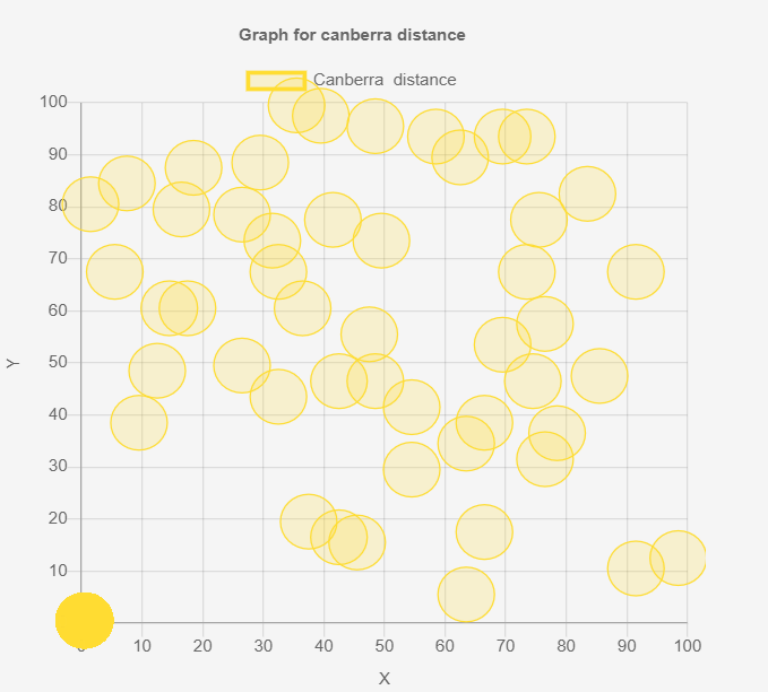


Cazul distantei Chebyshev este unul nefavorabil formandu-se doar doua clustere fixe la extremitati, din pricina modului in care se calculeaza distanta.

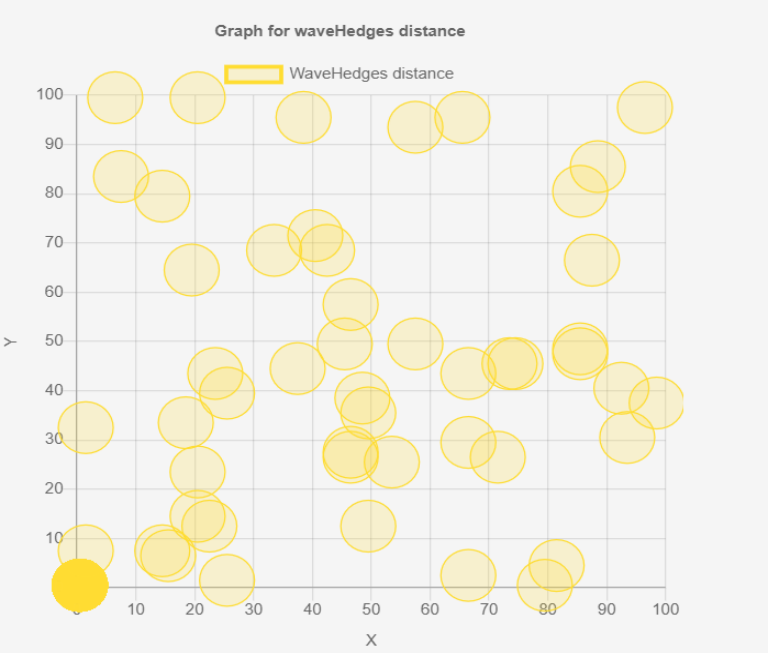
Urmatoarea distanta folosita a fost distanta statistica, rezultatele fiind destul de satfisfacatoare.

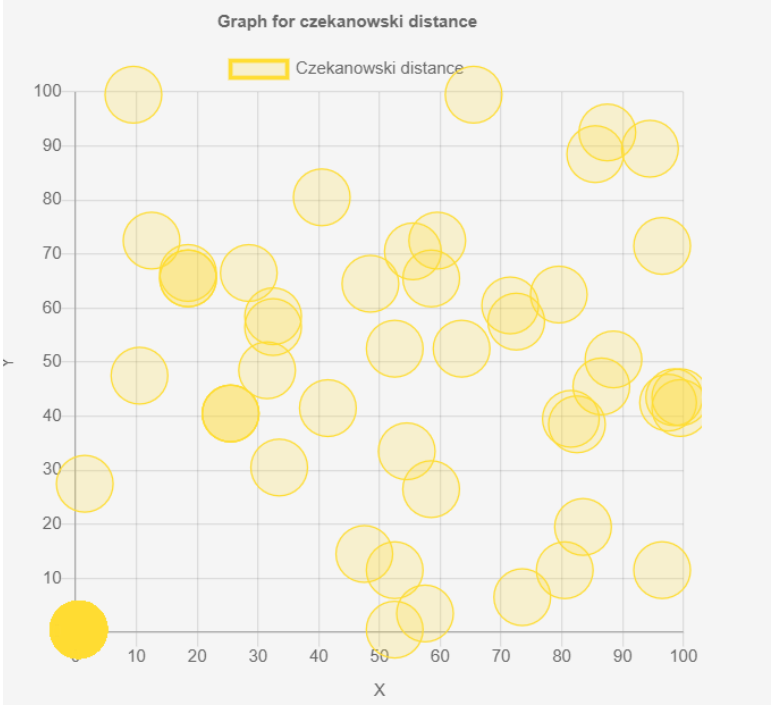


In continuare am folosit distanta Canberra insa cu mici sane de success deoarece aceasta este bazata pe distantele manhattan, ne mai fiind folosita semnificativ in invatare automata.



Urmatoarele doua distante sunt inrudite, amandoua facand parte din familia distantelor ce folosesc intersectia.





In finala distanta alesa pentru reteaua neuronala a fost cea Czeakanowski, din pricina distributiei datelor si buna clusterizare a acestora.

Astfel a fost creata reteaua Kohonen, cu ajutorul careia se vor oferi recomandari si preziceri.

Reteaua, optiunile pentru aceasta si datele din setul de antrenament au fost folosite o singura data, dupa care reteaua a fost salvata la nivelul serverlui sub format json, urmand ca la fiecare pornire a acestuia, reteaua sa fie reincarcata din fisier si nu inceperea procesului de la capat.

```
@Controller('reddit-users')
export class RedditUsersController {
    private kohonenNetwork = new KohonenNetwork();
    constructor(private readonly commandBus: CommandBus, private readonly queryBus: QueryBus) {
        this.kohonenNetwork.loadNetwork().subscribe(() => {
            console.log('Trained');
            sleeper(10000).then(() => {
                this.kohonenNetwork.trainNetwork();
            });
        });
    }
```

Exemplificarea folosirii API-ului celor de la Reddit si a crawler-ului.

Avand in vedere ca colectarea datelor reprezinta un pas de baza din aplicatia propusa, in continuare voi exemplifica modul in care sunt obtinute datele si API-ul corespunzator al siteului reddit.

Deoarece reddit considera ca datele precum comentariile sau posturile create ar trebui sa fie publice, nu a fost nevoie de urmarea protocolului OAuth pentru a obtine aceste date. Pentru utilizatori a fost folosit numai API-ul folosit de catre reddit. Datorita

volumului de date existent pe platforma, am cosiderat ca cele mai semnificative date se afla in ultimele 6 luni de activitate a utilizatorului. Astfel, cand un utilizator este creat, cele mai vechi comentarii si postari dateaza cu maxim 6 luni in urma momentului actual.

```
private numberOfMonthsToFetch: number = 6;
```

```
const boundaryDate = new Date();
boundaryDate.setMonth(boundaryDate.getMonth() - this.numberOfMonthsToFetch);
const unixDate: number = parseInt((boundaryDate.getTime() / 1000).toFixed(0), 10);
do {
    await sleeper().then(() => this.getOneBatchOfComments(
        redditCommentOwner, afterId).then((data) => { commentDate = data[0]; afterId = data[1]; }));
} while (commentDate > unixDate);
```

Aceste date nu sunt singurele ce pot fi colectate de catre aplicatie, fiind posibil ca, ulterior, dupa adaugarea unui user, comentariile sau postarile acestuia sa fie actualizate pana la momentul current.

Pentru datele ce tin de postarile de pe reddit este folosit atat crawlerul cat si API-ul.

Datele propriu-zise dintr-o postare sunt obtinute cu ajutorul apelurilor catre interfata propusa de catre reddit iar crawlerul incearca sa gaseasca si sa analizeze alte postari ce sunt poate accesibile din linkul curent.

Exemplificarea si argumentarea folosirii API-ului de la google.

Daca in ceea ce priveste invatarea automata in cardrul analizei utilizatorilor de reddit folosim o retea neuronala, pentru a extrage date ce tin de limbajul natural, vom folosi API-ul Google pentru.

Acest API foloseste pentru autorizare doar un singur key ce se instantiaza in cadrul consolei google.

Toate apelurile catre google folosesc aceasta cheie privata. Cele trei apeluri catre google sunt folosite pentru:

- Analiza entitatilor.
- Analizarea sentimentelor.
- Si clasificarea textului.

Datele sunt validate si inainte sa fie trimise si dupa ce au fost primite de la modulul de invatare automata a google-ului.

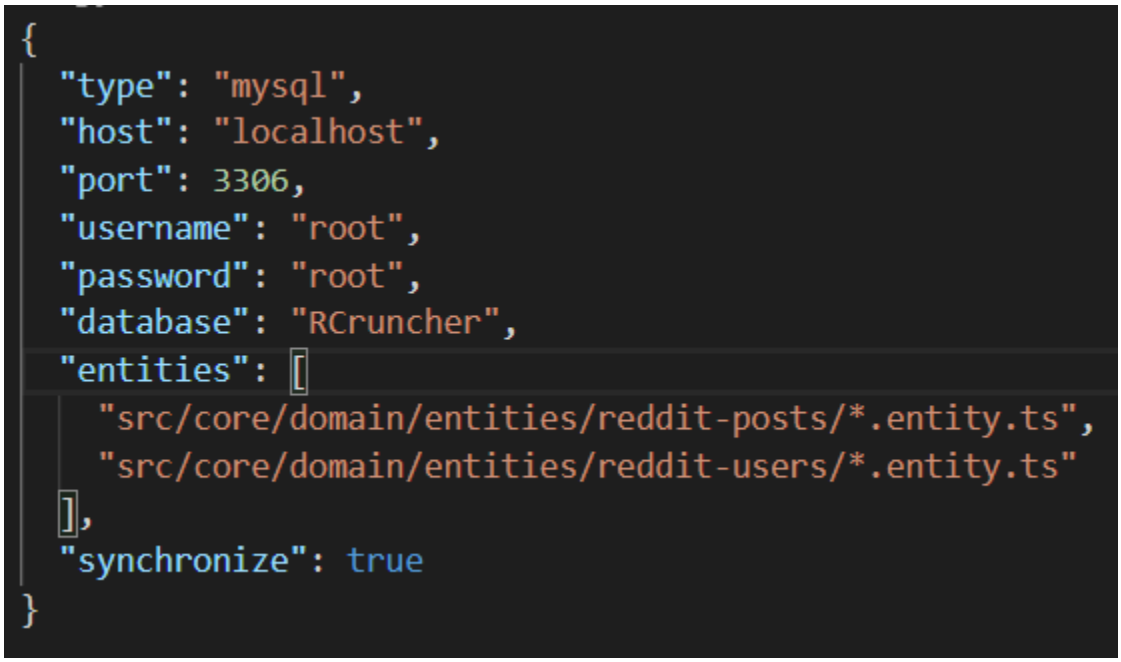
Baza de date

Baza de date este o baza de date relationala, mai exact MySQL si consta in 8 tabele ce vor fi mentionate sumar:

- reddit_user_entity, ce se ocupa cu stocarea utilizatorilor si datelor despre acestia.
- reddit_comment_entity, care stocheaza pentru fiecare utilizator comentariile colectate.
- reddit_topic_entity, in care se salveaza topicurile extrase din comentarii.
- user_subreddit_entity, se ocupa cu stocarea postarilor create de fiecare utilizator
- reddit_post_entity, in care se salveaza posturile cu datele asociate acestora
- google_natural_language_sentence, in care se salveaza datele primite de la google in legatura cu propozitile componente ale postarii
- google_natural_language_entity, stocheaza entitatile din postare, entitati primte de la google
- google_natural_language_category, in care se salveaza categoriile din care face parte o postare, categorii obtinute dupa apelarea modulului de limbaj natural al google-ului.

Conexiunea cu baza de date

Pentru conexiunea cu baza de date a fost folosit un object-relational mapping, si mai exact typeorm. Setarile cu care a fost creata conexiunea sunt urmatoarele:



```
{
  "type": "mysql",
  "host": "localhost",
  "port": 3306,
  "username": "root",
  "password": "root",
  "database": "RCruncher",
  "entities": [
    "src/core/domain/entities/reddit-posts/*.entity.ts",
    "src/core/domain/entities/reddit-users/*.entity.ts"
  ],
  "synchronize": true
}
```

Baza de date se numeste RCruncher iar tabelele sunt reprezentate de catre entitatile aflate in cele doua foldere: “entities/reddit-posts” si “entities/reddit-users”. Fiecare entitate are decoratorul de Entity si extinde BaseEntity, comportandu-se astfel, cand este nevoie, ca un repository pattern.

Clientul si arhitectura acestuia

Pentru partea de client, am optat sa folosesc arhitectura SPA(singe-page application) deoarece chiar daca volumul datelor este ridicat pe partea de server, pe partea de client trebuie doar reprezentate visual acestea.

Limbajul folsit este tot typescript, coincizand astfel cu cel de pe server. Frameworkul folosit este Angular, framework ce reprezinta o solutie viabila pentru aplicatii de tip SPA.

Astfel partea de client este formata din cateva pagini ce vor fi prezentate in continuare:

- Pagina de home, in care sunt prezentate statisticile aplicatiei.
- Pagina pentru retea, in care pot fi adaugati utilizatori si analizati
- Pagina pentru postari, unde este disponibila analiza si vizualizarea datelor ce tin de posturile reddit.

Concluzii

Bibliografie

- NestJs, https://docs.nestjs.com/
- Node.js, https://nodejs.org/en/docs/
- Retea neuronala, https://en.wikipedia.org/wiki/Self-organizing_map
- Machine learning, https://github.com/mljs
- TypeOrm, https://typeorm.io/#/
- Baza de date, https://www.mysql.com/
- AngularJS, https://docs.angularjs.org/api
- Documentatie distanta, http://www.naun.org/main/NAUN/ijmmas/mmmas-49.pdf
- Sabin Buraga, Proiectarea siturilor Web, DESIGN SI FUNCTONALITATE, Editura Polirom, mai 2005
- Liviu Ciortuz, Alina Munteanu, Elena Badarau, Exercitii de invatare automata, Editura Universitatii Alexandru Ioan Cuza, 2015