ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Кафедра ЕОМ

Атестаційна робота на тему: «Методи визначення текстової близкості фраз»

Виконала: ст. гр. СПм-20-2 Ляшова А.О.
Керівник: проф. каф. ЕОМ Фесенко Т.Г.

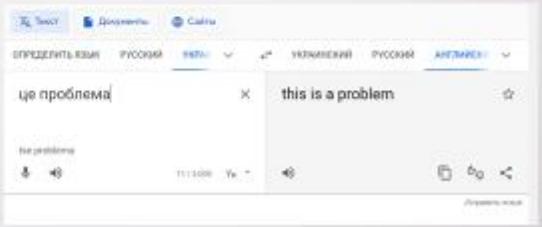
Мета доповіді

Метою доповіді є розробка смарт-бота для перевірки автентичності інформації, отриманої з новинних каналів, використовуючи методи визначення текстової близькості.

Поставлені задачі:

- реалізувати взаємодії системи з офіційними, достовірними джерелами;
- проаналізувати отриману інформацію;
- надати користувачу відсоток достовірності вхідної інформації.

Аналогічні системи



Актуальність поставленої задачі

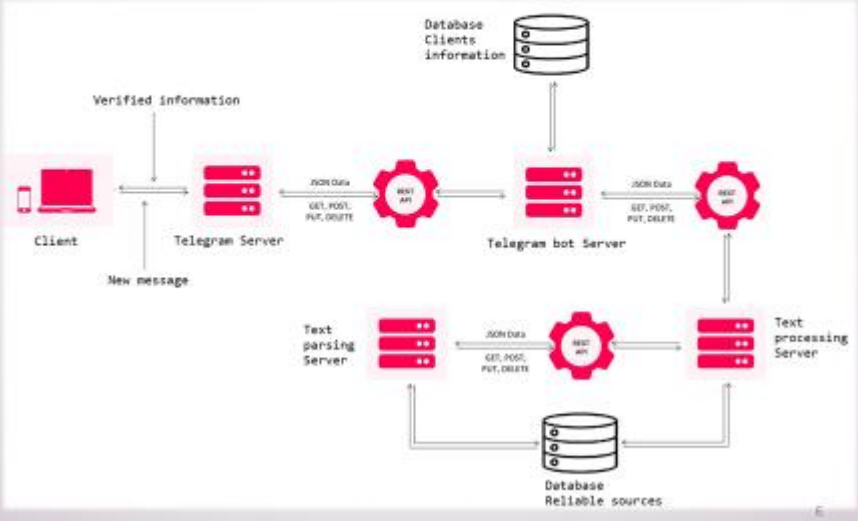
Найпоширенішими джерелами отримання інформації є Telegram канали. Але в наш час більшість публікують неправдиву інформацію та поширюють пропаганду. Люди навіть не замислюються про перевірку достовірності даних.

Саме тому запропонована робота буде розробка Telegram бота який, в свою чергу, будет допомогати користувачам Telegram каналів фільтрувати отриману інформацію.

Методи визначення близкості текстів

1. Метод порівняння текстів
2. Метод латентно-семантичного аналізу
3. Метод визначення подібності
4. Метод Дамерау-Левенштейна

Схема запропонованої системи



Метод Дамерау-Левенштейна

Відстань Дамерау-Левенштейна – це метрика визначення відстані між двома рядками.

Основними операціями прямого метода Дамерау-Левенштейна є:

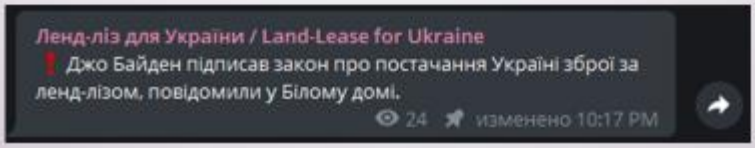
- вставка;
- заміна;
- видалення;
- перестановка.

Відстань між двома рядками — це мінімальна загальна кількість операцій, які перетворюють один рядок на інший.

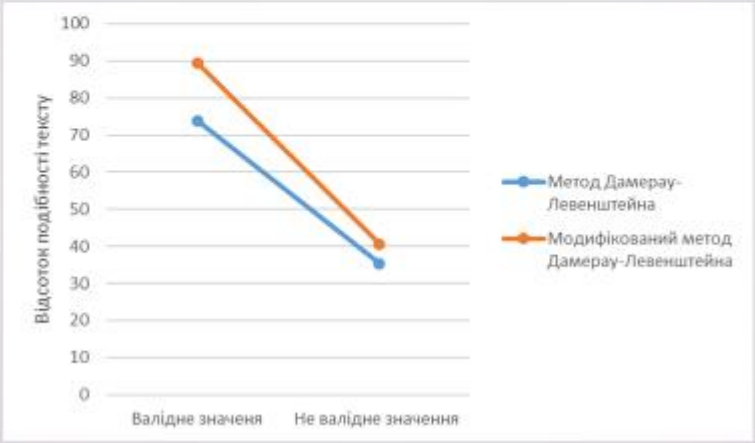
Блок схема модифікованного метода Дамерау-Левенштейна

Тестування системи

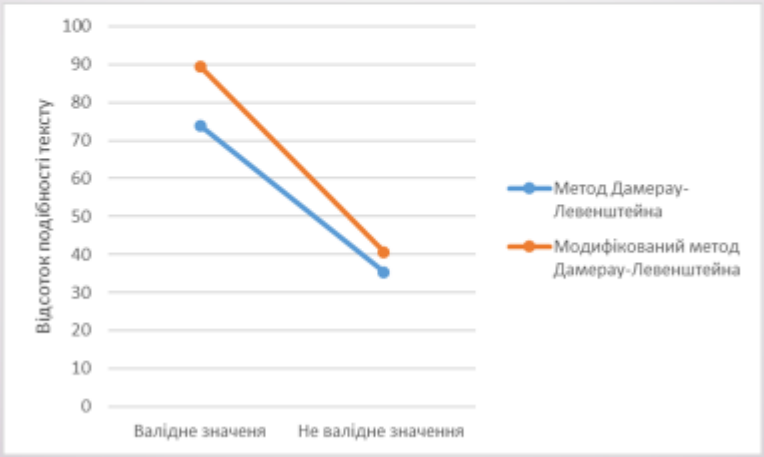
Новина з офішійного сайту (еталонне значення)	Президент США Джо Байден 9 травня має підписати закон про ленд-ліз для України. Аналогічна програма працювала вісімдесят років тому саме під час Другої світової війни.
Новина у каналі (вхідний текст)	Джо Байден підписав закон про постачання Україні зброї за ленд-лізом, повідомили у Білому домі.
Не валідне значення (вхідний текст)	Джо Байден відмовився підписувати закон про ленд-ліз.



Отримані результати Залежності відсотків подібності тексту



Отримані результати
Час витрачений на обробку тексту



11

Висновки

В даний час спостерігається велике зростання обсягів інформації, що зберігається в Інтернеті. При цьому досить важливим і поширеним є виявлення та усунення дезінформації. На основі методів визначення близкості текстів розробляєма система буде аналізувати та оцінювати вхідні дані. Після перевірки даних користувая отримає результат достовірності інформації.

Таким чином знизиться ризик отримання дезінформації, яка, в наш час, впливає на поведінку та дії людини.

12

Ляшова А.

Стаття у фаховому виданні:

- BarkovskaO.Yu. Прискорений алгоритм пошуку слів-образів у тексті з адаптивною декомпозицією вихідних даних / BarkovskaO.Yu., D.I. Pyvovarova, V.S. Serdechnyi, LiashovaA.A. // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2019. – Т. 4 (56). – С. 28-34. – doi:https://doi.org/10.26906/SUNZ.2019.4.028.

Тези доповіді:

- Барковська О. Ю., Ляшова А. О. Актуальність методів визначення текстової близькості для аналізу публікацій у новинних каналах / Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей 12-ої МНТК (online-режим). – 2022. – Т.1: секція 2. – с.43.


СИСТЕМИ УПРАВЛІННЯ, НАВІГАЦІЇ ТА ЗВ'ЯЗКУ


СУЧАСНІ НАПРЯМИ РОЗВИТКУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ УПРАВЛІННЯ

ДОДАТОК Б
ПРОГРАМНИЙ КОД

Б.1 Програмна частина

Б.1.1 Реалізація фільтрів для Telegram боту

```
package nure.ua.liashova.textProccessingBotApi
.Bot.botController.filters;


import nure.ua.liashova.textProccessingBotApi
.Bot.botController.Bot;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.User;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.mapper.MapperUser;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.repository.UserRepository;
import org.springframework.beans.factory.annotation
.Autowired;
import org.telegram.telegrambots.bots
.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.objects.Update;

public class CustomFilter {
    TelegramLongPollingBot bot = Bot.getInstance();


    MapperUser mapperUser=new MapperUser();

    UserRepository
userRepository=UserRepository.getInstance();

    public boolean filter(Update update) {

        return true;
    }
}

package nure.ua.liashova.textProccessingBotApi
.Bot.botController.filters;

import nure.ua.liashova.textProccessingBotApi
.Bot.botController.Message;
import nure.ua.liashova.textProccessingBotApi
```

```
.Bot.botController.User;
import org.telegram.telegrambots.meta.api.objects.Update;

public class Filter1 extends CustomFilter {

    public boolean filter(Update update) {

        User user = mapperUser.mapUpdateToUser(update);
        User saved = userRepository.getUserByUpdate(update);
        if (saved == null) {
            saved = userRepository.save(user);
        }
        saved.addMessages(new Message(update.getMessage()
                                              .getText()));
        userRepository.save(saved);
        return false;
    }
}
```

Б.1.2 Реалізація ініціалізації користувача для Telegram боту

```
package nure.ua.liashova.textProccessingBotApi
.Bot.botController.mapper;

import nure.ua.liashova.textProccessingBotApi
.Bot.botController.User;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.util.Status;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;

@Service
public class MapperUser {

    public User mapUpdateToUser(Update update) {
        User user = new User();
        user.setUserId(update.getMessage().getChatId());

user.setName(update.getMessage().getChat().getFirstName());
        user.setStatus(Status.UNREGISTER);
        return user;
    }
}

package nure.ua.liashova.textProccessingBotApi
.Bot.botController.repository;

import nure.ua.liashova.textProccessingBotApi
.Bot.botController.User;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;
```

```
import java.util.HashMap;
import java.util.Map;

@Service
public class UserRepository {
    static UserRepository instance;

    public static UserRepository getInstance() {
        if (instance == null) {
            instance = new UserRepository();
        }
        return instance;
    }

    public static Map<Long, User> users = new HashMap();

    public User getUserById(long id) {
        return users.get(id);
    }

    public User getUserByUpdate(Update update) {
        return users.get(update.getMessage().getChatId());
    }

    public User save(User user) {
        users.put(user.getUserId(), user);
        return user;
    }
}

package nure.ua.liashova.textProccessingBotApi
.Bot.botController;

import nure.ua.liashova.textProccessingBotApi
.Bot.botController.util.Status;
import org.telegram.telegrambots.meta.api.objects
.replykeyboard.ReplyKeyboardMarkup;

import java.util.ArrayList;
import java.util.List;

public class User {
    private long userId;
    private String name;
    private Status status = Status.UNREGISTER;
    private ReplyKeyboardMarkup keyboard = null;
    private List<Message> messages = new ArrayList<>();

    public long getUserId() {
        return userId;
    }
```

```
    public User setUserId(long userId) {
        this.userId = userId;
        return this;
    }

    public String getName() {
        return name;
    }

    public User setName(String name) {
        this.name = name;
        return this;
    }

    public Status getStatus() {
        return status;
    }

    public User setStatus(Status status) {
        this.status = status;
        return this;
    }

    public ReplyKeyboardMarkup getKeyboard() {
        return keyboard;
    }

    public User setKeyboard(ReplyKeyboardMarkup keyboard) {
        this.keyboard = keyboard;
        return this;
    }

    public List<Message> getMessages() {
        return messages;
    }

    public void addMessages(Message message) {
        messages.add(message);
    }

    @Override
    public String toString() {
        String res = "";
        for (Message message : messages) {
            res += message+System.lineSeparator();
        }
        return
                "userId=" + userId + "\n" +
                "name=" + name + "\n" +
                "status=" + status +"\n"+
                "messages=\n" + res ;
    }
}
```

Б.1.3 Реалізація Telegram бота

```
package nure.ua.liashova.textProccessingBotApi
.Bot.botController;

import nure.ua.liashova.textProccessingBotApi
.Bot.botController.filters.CustomFilter;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.listeners.CustomListener;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.mapper.MapperUser;
import nure.ua.liashova.textProccessingBotApi
.Bot.botController.repository.UserRepository;
import org.telegram.telegrambots.bots
.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods
.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions
.TelegramApiException;

import java.util.ArrayList;
import java.util.List;

public class Bot extends TelegramLongPollingBot {
    private static Bot instance;

    private    List<CustomListener>    listeners    =    new
ArrayList<>();
    private List<CustomFilter> filters = new ArrayList<>();

    public static Bot getInstance() {
        if (instance == null) {
            instance = new Bot();
        }
        return instance;
    }

    public void addFilter(CustomFilter filter) {
        this.filters.add(filter);
    }

    public void addListener(CustomListener listener) {
        this.listeners.add(listener);
    }

    private MapperUser mapperUser = new MapperUser();
    private      UserRepository      userRepository      =
UserRepository.getInstance();
```

```
        public void onUpdateReceived(Update update) {
            User user = mapperUser.mapUpdateToUser(update);
            if (userRepository.getUserByUpdate(update) == null)
{
                userRepository.save(user);
            }


            for (CustomFilter f : filters) {
                if (f.filter(update)) {
                    return;
                }
            }
            for (CustomListener listener : listeners) {
                listener.listen(update);
            }
        }

        public void sendMSG(SendMessage sendMessage, String id)
{
            try {
                sendMessage.setChatId(id);
                execute(sendMessage);
            } catch (TelegramApiException e) {
                e.printStackTrace();
            }
        }


        public String getBotUsername() {
            return "textProcessingForNewsBot";
        }

        public String getBotToken() {
            return
"5146106769:AAEuq3ESZ16cD1F3YtC_rJ9zB6YTUyAgSA8";
        }
    }

    package nure.ua.liashova.textProccessingBotApi
    .Bot.botController;

    import java.util.Date;

    public class Message {
        private  String message;
        private Date date;

        public Message(String message) {
            this.message = message;
            date=new Date();
        }
```

```
    @Override
    public String toString() {
        return date+" : "+message;
    }
}
```

Б.1.4 Реалізація методів обробки тексту

```
package nure.ua.liashova.textProccessingBotApi
.Bot.botController.textProcessing;

import org.apache.tomcat.util.buf.StringUtils;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.OptionalDouble;

public class Processing {

    public static OptionalDouble textProcessing(String inputText, String
referenceText) {
        List<String> inputList = Arrays.stream(inputText.split("\\."))
                                       .toList();
        List<String>                   referenceList                   =
Arrays.stream(referenceText.split("\\."))
                                           .toList();
        List<Double> proximityFactorList = new ArrayList<Double>();

        for (String inputSentence : inputList) {
            String    sameSentence    =    sameSentences(inputSentence,
referenceList);

            List<String>                  inputWords                   =
Arrays.stream(inputSentence.split(" "))
                                            .toList();
            List<String>               sameSentenceWords               =
Arrays.stream(sameSentence.split(" "))
                                                    .toList();

            List<String> processInputWords = stemming(inputWords);
            List<String>           processSameSentenceWords            =
stemming(sameSentenceWords);

            int sameWordCount = 0;

            for (String word1 : processInputWords) {
                for (String word2 : processSameSentenceWords) {
                    int  distance  =  DamerauLevenshteinDistance(word1,
word2);

                    if (distance < 3) {
                        sameWordCount++;
                        break;
                    }

                }
            }
```

```
                proximityFactorList.add(proximityFactor(inputWords.size(),
sameSentenceWords.size(), sameWordCount));
            }

            return proximityFactorList.stream()
                                      .toList()
                                      .stream()
                                      .mapToDouble(e -> e)
                                      .average();
        }

        public static String sameSentences(String sentence, List<String>
list) {
            List<String> sentenceList = Arrays.stream(sentence.split(" "))
                                              .toList();
            String returnSentences = null;
            int numberOfTheSameWords = 0;

            for (String s : list) {
                List<String> sList = Arrays.stream(s.split(" "))
                                           .toList();
                int count = 0;

                for (String i : sentenceList) {
                    for (String j : sList) {
                        if (i.equals(j)) {
                            count++;
                            break;
                        }
                    }
                }

                if (numberOfTheSameWords < count) {
                    numberOfTheSameWords = count;
                    returnSentences = s;
                }
            }

            return returnSentences;
        }

        public static int DamerauLevenshteinDistance(String str1, String
str2) {
            int[] Di_1 = new int[str2.length() + 1];
            int[] Di = new int[str2.length() + 1];

            for (int j = 0; j <= str2.length(); j++) {
                Di[j] = j; // (i == 0)
            }

            for (int i = 1; i <= str1.length(); i++) {
                System.arraycopy(Di, 0, Di_1, 0, Di_1.length);

                Di[0] = i; // (j == 0)
                for (int j = 1; j <= str2.length(); j++) {
                    int cost = (str1.charAt(i - 1) != str2.charAt(j - 1)) ?
1 : 0;
                    Di[j] = min(
                            Di_1[j] + 1,
                            Di[j - 1] + 1,
                            Di_1[j - 1] + cost
                    );
                }
            }
```

```
            return Di[Di.length - 1];
        }

        private static int min(int n1, int n2, int n3) {
            return Math.min(Math.min(n1, n2), n3);
        }

        public static double proximityFactor(double a, double b, double c) {
            return c / (a + b - c);
        }



        private static List<String> stemming(List<String> inputList) {
            List<String> outputList = new ArrayList<>();
            String[] endingList = {"ов", "ев", "єв", "им", "ом", "ов", "ин",
"ін", "у", "ю", "ові", "еві", "єві", "і", "ї"};
            String[] prefixesList = {"з", "с", "без", "від", "од", "між",
"над", "об", "під", "перед", "понад", "пред", "роз"
                    , "через", "пре", "при", "прі"};

            for (String word : inputList) {
                String outputValue = word;

                for (String end : endingList){
                    if(word.endsWith(end)){
                        outputValue = outputValue.replace(end, "");
                        break;
                    }
                }

                for(String prefix : prefixesList){
                    if(word.startsWith(prefix)){
                        outputValue = outputValue.replace(prefix, "");
                        break;
                    }
                }

                outputList.add(outputValue);
            }

            return outputList;
        }

        public static void main(String[] args) {
            String inputText = "aa bc. cd.";
            String referenceText = "cc aa bc. cd.";

            textProcessing(inputText, referenceText);
        }
    }
```