|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**        Дисципліна  **«Емпіричні методи програмної інженерії»**      **ДОМАШНЄ ЗАВДАННЯ**  **“**Обробка неструктурованих даних як етап емпіричних досліджень програмного забезпечення**”** | | | |
| **Виконав:** | Гоша Д. О | **Перевірила**: | Юрчук Ірина Аркадіївна |
| Група | ІПЗ-23 | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2022 | | | |

Мета: Практичне засвоєння методів, прийомів та навиків отриманих під час вивчення дисципліни емпіричні методи програмної інженерії на прикладі проведення обробки неструктурованих даних, які супроводжують процес створення та роботи програмного забезпечення; навчитися обробляти неструктуровані дані

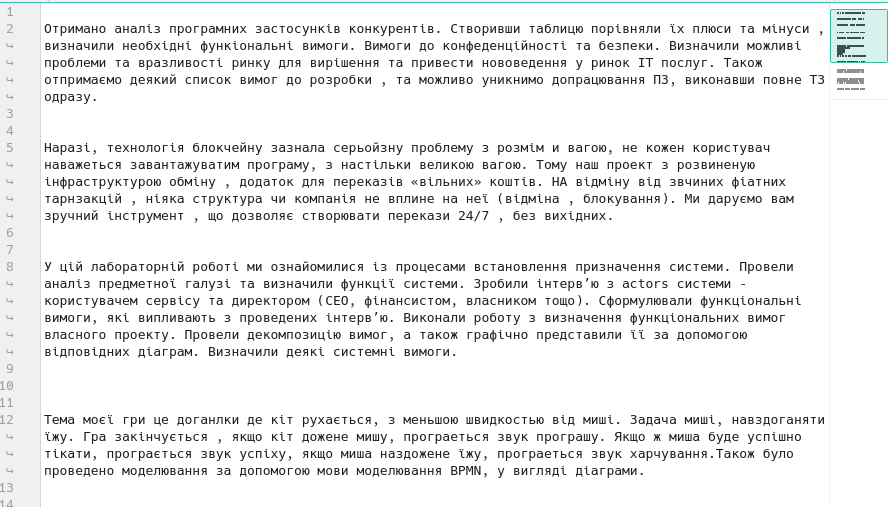
**Завдання**

1. Зібрати неструктуровані дані: текстові файли, веб-сайти, лог-файли, e-mails, які супроводжують процес проектування, створення або рефакторингу ПЗ. Рекомендована кількість 20-30 файлів;
2. Проаналізувати отримані дані;
3. На основі проведеного аналізу провести їх передобробку та відбір. Вибір алгоритмів та їх кроки обґрунтувати;
4. Представити структуровані дані, які можуть бути використанні для подальшого аналізу (з метою прогнозування, класифікації і т.п.);
5. Зробити висновки (у вигляді рекомендації) про можливість подальшого використання отриманих даних.

# Виконання

**Опис зібраних даних**

Неструктуровані дані представляють будь-які дані, які не мають впізнаваної структури. Він неорганізований і сирий і може бути не текстовим або текстовим. Наприклад, електронна пошта - це прекрасна ілюстрація неструктурованих текстових даних. Вона включає час, дату, дані одержувача та відправника, тему тощо, але тіло електронної пошти залишається неструктурованим. Неструктуровані дані також можуть бути ідентифіковані як слабко структуровані дані, де джерела даних включають структуру, але не всі дані в наборі даних відповідають одній структурі. Для аналізу мною були вибрані 10 файлів з висновками до лабораторних робіт з дисципліни аналіз вимог до програмного забезпечення.



Текстові дані з файлів були зібрані та обʼєднані у текстовий файл. Це можна вважати з перший етап обробки – структуризація хаотичних даних.

**Формалізовані етапи обробки та описи алгоритмів**

**Токенізація**

Токенізація - це акт розбиття послідовності рядків на частини, такі як слова, ключові слова, фрази, символи та інші елементи, що називаються лексемами. Маркерами можуть бути окремі слова, фрази або навіть цілі речення. У процесі токенізації деякі символи, як розділові знаки, відкидаються. Маркери стають вхідним сигналом для іншого процесу, такого як розбір і пошук тексту.

Токенізація в основному покладається на просту евристику, щоб розділити лексеми, виконавши кілька кроків:

* Маркери або слова розділені пробілом, розділовими знаками або розривами рядків
* Білий пробіл або розділові знаки можуть бути або не включатися залежно від потреби
* Усі символи в суміжних рядках є частиною маркера.

Токени можуть складатися з усіх альфа-символів, буквено-цифрових символів або лише числових символів.

Самі жетони також можуть бути роздільниками. Наприклад, у більшості мов програмування ідентифікатори можуть бути розміщені разом з арифметичними операторами без пробілів. Хоча здається, що це виглядатиме як одне слово чи лексема, граматика мови насправді розглядає математичний оператор (маркер) як роздільник, тому навіть коли кілька лексем зібрано разом, їх все одно можна розділити за допомогою математичного оператор.

Створимо спеціальний додаток для токенізації слів на мові програмування python використовуючи бібліотеку Stanza.

Наведемо приклад програмного коду і результату нижче:

import stanza

nlp = stanza.Pipeline(lang='uk', processors='tokenize')

doc = nlp('Отримано аналіз програмних застосунків конкурентів. Створивши таблицю порівняли їх плюси та мінуси , визначили необхідні функіональні вимоги')

for i, sentence in enumerate(doc.sentences):

print(f'====== Речення номер {i+1} токену =======')

print(\*[f'id: {token.id}\tлексема: {token.text}' for token in sentence.tokens], sep='\n')

====== Речення номер 1 токену =======

id: (1,) лексема: Отримано

id: (2,) лексема: аналіз

id: (3,) лексема: програмних

id: (4,) лексема: застосунків

id: (5,) лексема: конкурентів

====== Речення номер 2 токену =======

id: (1,) лексема: Створивши

id: (2,) лексема: таблицю

id: (3,) лексема: порівняли

id: (4,) лексема: їх

id: (5,) лексема: плюси

id: (6,) лексема: та

id: (7,) лексема: мінуси

id: (9,) лексема: визначили

id: (10,) лексема: необхідні

id: (11,) лексема: функіональні

id: (12,) лексема: вимоги

Таким чином було токенізовано всі 10 проектів, результати яких було записано до спеціального текстового файлу. У зручному форматі, для майбутніх типів обробок.

**Видалення стоп слів**

Процес перетворення даних у те, що може зрозуміти комп’ютер, називається попередньою обробкою. Однією з основних форм попередньої обробки є фільтрація непотрібних даних. При обробці природної мови непотрібні слова (дані) називаються стоп-словами.

Стоп-слово — це часто вживане слово (наприклад, «та», «як», «а», «але»), яке пошукова система була запрограмована на ігнорування, як під час індексації записів для пошуку, так і під час їх отримання. як результат пошукового запиту. Ми б не хотіли, щоб ці слова займали місце в нашій базі даних або забирали дорогоцінний час обробки. Для цього ми можемо легко видалити їх, зберігши список слів, які ви вважаєте зупинковими словами.

Стоп-слова були відібрані з цього репозиторію <https://github.com/skupriienko/Ukrainian-Stopwords.> Загальна кількість складає 1983 слова. Було написано наступний застосунок, точніше модернізовано попередній, для видалення стоп слів.

import pandas as pd

from nltk.tokenize import word\_tokenize

example\_sent = """Отримано аналіз програмних застосунків конкурентів. Створивши таблицю порівняли їх плюси та мінуси , визначили необхідні функіональні вимоги. Вимоги до конфеденційності та безпеки. Визначили можливі проблеми та вразливості ринку для вирішення та привести нововедення у ринок IT послуг. Також отпримаємо деякий список вимог до розробки , та можливо уникнимо допрацювання ПЗ, виконавши повне ТЗ одразу."""

stopwords\_ua = pd.read\_csv("stopwords\_ua.txt", header=None, names=['stopwords'])

stop\_words = list(stopwords\_ua.stopwords)

word\_tokens = word\_tokenize(example\_sent)

filtered\_sentence = [w for w in word\_tokens if not w.lower() in stop\_words]

filtered\_sentence = []

for w in word\_tokens:

if w not in stop\_words:

filtered\_sentence.append(w)

print(word\_tokens)

print(filtered\_sentence)

Приклад результату роботи

До обробки.(Токенізований список)

['Отримано', 'аналіз', 'програмних', 'застосунків', 'конкурентів', '.', 'Створивши', 'таблицю', 'порівняли', 'їх', 'плюси', 'та', 'мінуси', ',', 'визначили', 'необхідні', 'функіональні', 'вимоги', '.', 'Вимоги', 'до', 'конфеденційності', 'та', 'безпеки', '.', 'Визначили', 'можливі', 'проблеми', 'та', 'вразливості', 'ринку', 'для', 'вирішення', 'та', 'привести', 'нововедення', 'у', 'ринок', 'IT', 'послуг', '.', 'Також', 'отпримаємо', 'деякий', 'список', 'вимог', 'до', 'розробки', ',', 'та', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

Після обробки.(Видалені стоп-слова)

['Отримано', 'аналіз', 'програмних', 'застосунків', 'конкурентів', '.', 'Створивши', 'таблицю', 'порівняли', 'плюси', 'мінуси', ',', 'визначили', 'необхідні', 'функіональні', 'вимоги', '.', 'Вимоги', 'конфеденційності', 'безпеки', '.', 'Визначили', 'можливі', 'проблеми', 'вразливості', 'ринку', 'вирішення', 'привести', 'нововедення', 'ринок', 'IT', 'послуг', '.', 'Також', 'отпримаємо', 'список', 'вимог', 'розробки', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

Лематизація

Лемматизація – це процес об’єднання різних форм слів, щоб їх можна було проаналізувати як єдиний елемент. Лемматизація подібна до стемінгу, але привносить контекст до слів. Таким чином, він зв’язує слова зі схожими значеннями в одне слово. Попередня обробка тексту включає як стеммінг, так і лемматизацію. Часто люди плутають ці два терміни.Багато хто зовсім не розрізняє їх. Насправді, лемматизація є кращою за стеммінг, оскільки лемматизація виконує морфологічний аналіз слів. Застосування лемматизації:

* Використовується в комплексних пошукових системах, таких як пошукові системи.
* Використовується в компактному індексуванні

Однією з основних відмінностей від стеммінгу є те, що Лематизація бере частину параметра мови, «pos». Якщо не вказано, за замовчуванням буде «іменник». Нижче наведено реалізація слів лемматизації за допомогою Simplemma:

import simplemma

from simplemma import text\_lemmatizer

langdata = simplemma.load\_data('uk')

Token\_text = ''

# Python program to convert a list to string

# Function to convert

def listToString(s):

# initialize an empty string

str1 = ""

# traverse in the string

for ele in s:

str1 += ' '+ele

# return string

return str1

print(text\_lemmatizer(listToString(Token\_text), langdata))

Приклад результату роботи

До обробки

['Отримано', 'аналіз', 'програмних', 'застосунків', 'конкурентів', '.', 'Створивши', 'таблицю', 'порівняли', 'плюси', 'мінуси', ',', 'визначили', 'необхідні', 'функіональні', 'вимоги', '.', 'Вимоги', 'конфеденційності', 'безпеки', '.', 'Визначили', 'можливі', 'проблеми', 'вразливості', 'ринку', 'вирішення', 'привести', 'нововедення', 'ринок', 'IT', 'послуг', '.', 'Також', 'отпримаємо', 'список', 'вимог', 'розробки', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

Після обробки

['отримати', 'аналіз', 'програмний', 'застосунків', 'конкурент', '.', 'створити', 'таблиця', 'порівняти', 'плюс', 'мінус', ',', 'визначити', 'необхідний', 'функіональні', 'вимога', '.', 'вимога', 'конфеденційності', 'безпека', '.', 'визначити', 'можливий', 'проблема', 'вразливості', 'ринок', 'вирішення', 'привести', 'нововедення', 'ринок', 'IT', 'послуга', '.', 'Також', 'отпримаємо', 'список', 'вимога', 'розробка', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконати', 'повня', 'ТЗ', 'одразу', '.']

**Стеммінг**

Стеммінг – це процес утворення морфологічних варіантів кореня/основи слова. Програми стемминга зазвичай називають алгоритмами стеммінгу або стеммерами. Алгоритм створення основ зводить слова «шоколад», «шоколадний», «шоколадний» до кореневого слова, «шоколад» і «вилучення», «здобуто», «здобуває» зводяться до основи «отримати». Стеммінг це важлива частина процесу обробці неструктурованих даних природної мови. Вхідні дані для stemmer є лексемними словами.   
import pymorphy2

morph = pymorphy2.MorphAnalyzer(lang='uk')

Token\_text = ['Отримано', 'аналіз', 'програмних', 'застосунків', 'конкурентів', '.', 'Створивши', 'таблицю', 'порівняли', 'плюси', 'мінуси', ',', 'визначили', 'необхідні', 'функіональні', 'вимоги', '.', 'Вимоги', 'конфеденційності', 'безпеки', '.', 'Визначили', 'можливі', 'проблеми', 'вразливості', 'ринку', 'вирішення', 'привести', 'нововедення', 'ринок', 'IT', 'послуг', '.', 'Також', 'отпримаємо', 'список', 'вимог', 'розробки', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

for ele in Token\_text:

p = morph.parse(ele)[0]

print(p.normal\_form, end=' ')

Приклад результату роботи

До обробки

['Отримано', 'аналіз', 'програмних', 'застосунків', 'конкурентів', '.', 'Створивши', 'таблицю', 'порівняли', 'плюси', 'мінуси', ',', 'визначили', 'необхідні', 'функіональні', 'вимоги', '.', 'Вимоги', 'конфеденційності', 'безпеки', '.', 'Визначили', 'можливі', 'проблеми', 'вразливості', 'ринку', 'вирішення', 'привести', 'нововедення', 'ринок', 'IT', 'послуг', '.', 'Також', 'отпримаємо', 'список', 'вимог', 'розробки', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

Після обробки

отримати аналіз програмний застосунок конкурент . створивши таблиця порівняти плюс мінус , визначити необхідний функіональний вимога . вимога конфеденційність безпека . визначити можливий проблема вразливість ринок вирішення привести нововедення ринок it послуга . також отпримати список вимога розробка , можливо уникнити допрацювання пз , виконавши повня тз одразу .

**Розрахунки у вигляді таблиць**

TF-IDF (від англ. TF - term frequency, IDF - inverse document frequency) - статистична міра, що використовується для оцінки важливості слова в контексті документа, що є частиною колекції документів або корпусу. Вага деякого слова пропорційна частоті вживання цього слова в документі і обернено пропорційна частоті вживання слова в усіх документах колекції.

Міра TF-IDF часто використовується в задачах аналізу текстів та інформаційного пошуку, наприклад, як один із критеріїв релевантності документа пошуковому запиту, при розрахунку міри близькості документів під час кластеризації.  
TF (term frequency – частота слова) – відношення числа входжень деякого слова до загального числа слів документа. Таким чином, оцінюється важливість слова ti у межах окремого документа.

де nt є число входжень слова t у документ, а знаменнику - загальна кількість слів у цьому документі.

IDF (inverse document frequency – зворотна частота документа) – інверсія частоти, з якою деяке слово зустрічається в документах колекції. Основоположником цієї концепції є Карен Спарк Джонс. Облік IDF зменшує вагу широковживаних слів. Для кожного унікального слова в межах конкретної колекції документів існує лише значення IDF.

де

* - Число документів у колекції;
* — кількість документів з колекції {\displaystyle D}D, у яких зустрічається t (коли ni != 0).

Вибір основи логарифму у формулі немає значення, оскільки зміна основи призводить до зміни ваги кожного слова на постійний множник, що не впливає на співвідношення ваг. Таким чином, міра TF-IDF є твором двох співмножників:



**Структуровані дані та висновки**

**Висновки**

Таким чином, задача оцінки якості програмних проектів поки що не отримала прийнятного розв’язку у сучасній науці. Існуючі поширені підходи на основі метрик нерідко призводять до результатів, які погано узгоджуються між собою та є достатньо складними для інтерпретації і прийняття рішень на їх основі. Академічні проекти дозволяють по-новому підійти до вирішення проблеми, водночас поки що не придатні для масового практичного застосування в основному з причин як своєї новизни, так і незавершеності концепцій, використаних для представлення показників якості по проектам. Подальші дослідження у цьому напрямку, на мій погляд, мають бути зосереджені на тому, щоб привести існуючу базу метрик програмного коду до такого вигляду, у якому вони могли б слугувати реальною опорою для прийняття рішень керівниками програмних проектів, використовуючи комплексні і узгоджені між собою показники. Використання природно зрозумілих людині аналогій (структура міста, вигляд дерева та ін.) допомогло б подати інформацію більш наглядно, розширити коло людей, які могли нею оперувати при реалізації проекту.

У останній лабораторній роботі 6 були здобуті навички з рефакторингу власної роботи (власного проекту), написаної раніше на C# в середовищі Visual Studio.