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

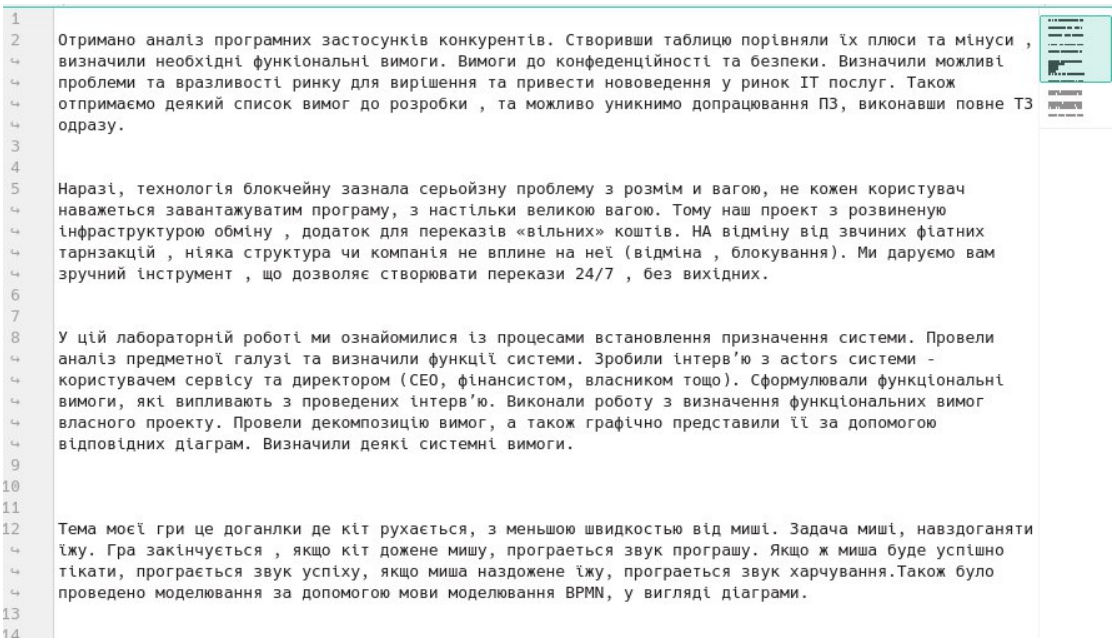
Завдання

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.



```
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.


Наразі, технологія блокчейну зазнала серьойзну проблему з розмім и вагою, не кожен користувач
наважеться завантажуватим програму, з настільки великою вагою. Тому наш проект з розвиненую
інфраструктурою обміну , додаток для переказів «вільних» коштів. НА відміну від звчиних фіатних
тарнзакцій , ніяка структура чи компанія не вплине на неї (відміна , блокування). Ми даруємо вам
зручний інструмент , що дозволяє створювати перекази 24/7 , без вихідних.


У цій лабораторній роботі ми ознайомилися із процесами встановлення призначення системи. Провели
аналіз предметної галузі та визначили функції системи. Зробили інтерв'ю з actors системи -
користувачем сервісу та директором (CEO, фінансистом, власником тощо). Сформулювали функціональні
вимоги, які випливають з проведених інтерв'ю. Виконали роботу з визначення функціональних вимог
власного проекту. Провели декомпозицію вимог, а також графічно представили її за допомогою
відповідних діаграм. Визначили деякі системні вимоги.



Тема моєї гри це доганлки де кіт рухається, з меньшою швидкостью від миші. Задача миші, навздоганяти
їжу. Гра закінчується , якщо кіт дожене мишу, програється звук програшу. Якщо ж миша буде успішно
тікати, програється звук успіху, якщо миша наздожене їжу, програється звук харчування.Також було
проведено моделювання за допомогою мови моделювання BPMN, у вигляді діаграми.

```

Текстові дані з файлів були зібрані та об'єднані 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.
Ми переконалися у важливість цього завдання в на будь-якому етапі або проекті НЛП, а також реалізували його за допомогою Python і Stanza для відстеження. Здається, що це проста тема, але як тільки ви вникнете в деталі кожної моделі токенізатора, помітите, що вона насправді досить складна.

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.](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 ІТ 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', 'ІТ', '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', 'ІТ', 'послуг', '.', 'Також', 'отпримаємо', 'список', 'вимог', 'розробки', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконавши', 'повне', 'ТЗ', 'одразу', '.']

У цій пункті використано різні бібліотеки, які можна використовувати для видалення стоп-слів із документу в Python. Також додано або видалити стоп-слова зі списків стандартних стоп-слів, наданих різними бібліотеками. Таким чином ми видалили стоп слова у всіх поданих документах.

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', 'ІТ', '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', 'ІТ', 'послуга', '.', 'Також', 'отпримаємо', 'список', 'вимога', 'розробка', ',', 'можливо', 'уникнимо', 'допрацювання', 'ПЗ', ',', 'виконати', 'повня', 'ТЗ', 'одразу', '.']

Можна зробити висновок, що замість використання стемера, можна використовувати лемматизатор , бібліотеку від Simplema, яка виконує повний морфологічний аналіз, щоб точно визначити лему для кожного слова. Виконання повного морфологічного аналізу дає щонайбільше дуже скромні переваги для пошуку. Важко сказати , тому що будь-яка форма нормалізації, як правило, не покращує ефективність пошуку інформації українською мовою в сукупності - принаймні не дуже сильно. Хоча для деяких запитів це дуже допомагає, для інших випадків, як правило, погіршує продуктивність. Стеммінг збільшує запам’ятовування, завдаючи шкоди точності.

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
отриман аналіз програмн застосунк конкурент . створ таблиц порівнял плю мін , визначил необхідн функіональн вимог . вимог конфеденційност безпек . визначил можлив проблем вразливост ринк вирішенн привест нововеденн ринок it послуг . також отпримаєм список вимог розробк , можлив уникним допрацюванн пз , викона повн тз одраз .

У цій частині ми досліджували створення коренів, практику визначення та вилучення основи слова за допомогою правил та евристики. Стімінг зменшує кількість текстових даних, що корисно під час навчання моделей, але ціною викидання частини слова ми втрачаємо сутність інформації.

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 t_i у межах окремого документа.

$$tf(t,d) = \frac{n_t}{\sum_k \binom{n}{k}}$$

де n_t 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.

$$idf(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}$$

де

- $|D|$ - Число документів у колекції;
- $\{d_i \in D \mid t \in d_i\}$ — кількість документів з колекції D, у яких зустрічається t (коли n_i != 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:

$$tf - idf(t,d,D) = tf(t,d) \times idf(t,D)$$

Отже створимо наступний метод для обрахунку TFIDF

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

Отримаємо ексель файл з усіма входженнями у вигляді стовпців та значення – документ у вигляді рядків.

Нижче наведемо приклад неструктурованих даних у таблиці.

	аспект	брати	роведени	формат	зір	блок	аналізова	метрика
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0,04428	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0,047971	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0,100112	0	0
7	0	0	0	0	0	0	0	0
8	0,024759	0	0	0	0,024759	0	0	0,123795
9	0	0	0	0	0	0	0	0

Структуровані дані та висновки

Заключним етапом структуризації даних буде аналіз всіх входжень на кількість повторювань у документі TF та кількості документів, поділених на суму документів, які містять слово умовне слово. Обернена частота даних визначає вагу рідкісних слів у всіх документах. Тобто величина IDF.

У першій таблиці зібрані максимально число входжень слів у всіх документах. На основі цих числових показників ми можемо зробити висновки про загальну тему зібраних файлів.

Зеленим кольором помічені числа, що найчастіше зустрічаються у певних файлах.

	навичка	аналіз	вимога	діаграма	провести	особливість	проект	функціональний	робот	лабораторний
0	0	0,0165035	0,04951051	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0,0111049	0	0	0
2	0	0,01332975	0,05331901	0,013329753	0,035241951	0	0,00982357	0,026659507	0,01371827	0,006859134
3	0	0	0	0,016906029	0,022348554	0	0	0	0	0
4	0,01908939	0	0,07220283	0	0	0	0	0,028881133	0,00743073	0,007430728
5	0,02036202	0,04620981	0	0,015403271	0	0,02036202	0,01135168	0	0,00792611	0,00792611
6	0	0,01506842	0	0,060273668	0,019919364	0,01991936	0	0,015068417	0,0077538	0,007753803
7	0,00872658	0,0066014	0,07261542	0	0	0	0,00486501	0	0,01358762	0,006793808
8	0,00985259	0	0,02235959	0	0	0,00985259	0,00549275	0,007453195	0,00767043	0,003835214
9	0	0	0,09704061	0	0,018325815	0	0,02043302	0,027725887	0,0071335	0,007133499

У наступній таблиці показані аномальні співпадіня уже у певних файлах. Тобто слово, що частіше за інші зустрічається у документі. На основі цих слів ми можемо виділити основну тему документу або лабораторної роботи. Але вже з меншою вірогідністю.

	ринок	відміна	система	миша	забезпечення	метода	idef0	тестування	метрика	створити
0	0,10964691	0	0	0	0	0	0	0	0	0
1	0	0,1001124	0	0	0	0	0	0	0	0
2	0	0	0,09285219	0	0	0	0	0	0	0
3	0	0	0	0,28080306	0	0	0	0	0	0
4	0	0	0	0	0,10058987	0	0	0	0	0
5	0	0	0	0	0	0,10729586	0	0	0	0
6	0	0	0,03498778	0	0	0	0,20022479	0	0	0
7	0	0	0,03065596	0	0	0	0	0,109646909	0	0
8	0	0	0	0	0,017305784	0	0	0,024758979	0,1237949	0
9	0	0	0	0	0	0	0	0	0	0,184206807

Отже ми зрозуміли, як працює TF-IDF, що дало змогу краще зрозуміти, як функціонують алгоритми машинного навчання. У той час як алгоритми машинного навчання традиційно краще працюють з числами, алгоритми TF-IDF допомагають їм розшифровувати слова, призначаючи їм числове значення або вектор. Це було революційним рішенням для машинного навчання, особливо в областях, пов’язаних з НЛП, таких як аналіз тексту. При аналізі тексту за допомогою машинного навчання алгоритми TF-IDF допомагають сортувати дані за категоріями, а також витягувати ключові слова. Це означає, що прості монотонні завдання, як-от додавання тегів у службу підтримки або рядків зворотного зв’язку та введення даних, можна виконати за секунди. Саме за цим алгоритмом працюють пошукові системи такі як Google.Вони можуть надавати інформацію, пов’язану з вашим пошуком, за лічені секунди. Векторизація тексту перетворює текст всередині документів у числа, тому алгоритми TF-IDF можуть ранжувати статті в порядку релевантності.

Висновки

Алгоритми не працюють із «сирими» даними. Більшість процесу — підготовка тексту чи промови, перетворення в вид, доступний сприйняття комп'ютером.

Очищення. З тексту видаляються непотрібні для машини дані. Це більшість знаків пунктуації, особливі символи, дужки, теги та ін. Деякі символи можуть бути значущими у конкретних випадках. Наприклад, у тексті про економіку символи валют мають сенс.

Препроцесинг. Далі настає великий етап попередньої обробки – препроцессингу. Це приведення інформації до виду, в якому вона зрозуміліша алгоритму. Популярні методи препроцессингу: приведення символів одного регістру, щоб усі слова були написані з маленької літери; токенізація — розбиття тексту на токени. Так називають окремі компоненти – слова, речення чи фрази; тегування частин мови - визначення частин мови в кожному реченні для застосування граматичних правил; лематизація та стеммінг - приведення слів до єдиної форми. Стемінг грубіший, він обрізає суфікси і залишає коріння. Лематизація - приведення слів до початкових словоформ, часто з урахуванням контексту; видалення стоп-слів - артиклів, вигуків та ін; спелл-чекінг - автокорекція слів, написаних неправильно. Методи вибирають відповідно до завдання.

Векторизація. Після попередньої обробки на виході виходить набір підготовлених слів. Але алгоритми працюють із числовими даними, а чи не з чистим текстом. Тому з вхідної інформації створюють вектори - представляють її як набір числових значень. Популярні варіанти векторизації - "мішок слів" і "мішок N-грам". У "мішку слів" слова кодуються цифри. Враховується лише кількість слова в тексті, а не їхнє розташування та контекст. N-грами – це групи із N слів. Алгоритм наповнює «мішок» не окремими словами зі своїми частотою, а групами кілька слів, і це допомагає визначити контекст. Використання алгоритмів машинного навчання. За допомогою векторизації можна оцінити, як часто в тексті зустрічаються слова. Але більшість актуальних завдань складніше, ніж визначення частоти — тут потрібні просунуті алгоритми машинного навчання. Залежно від типу конкретного завдання створюється та налаштовується своя окрема модель. Алгоритми обробляють, аналізують та розпізнають вхідні дані, роблять на їх основі висновки. Це цікавий і складний процес, в якому багато математики та теорії ймовірностей.

Корисні посилання

1. https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76
2. https://pymorphy2.readthedocs.io/en/stable/user/guide.html
3. https://github.com/amakukha/stemmers_ukrainian
   https://github.com/Desklop/Uk_Stemmer
4. https://pypi.org/project/simplemma/
5. https://www.geeksforgeeks.org/introduction-to-stemming/?ref=lbp
6. https://github.com/nltk/nltk/wiki
7. https://uk.waldorf-am-see.org/104418-stopword-removal-with-nltk-THQRAK
8. https://stanfordnlp.github.io/stanza/installation_usage.html
9. http://www.tfidf.com/#:~:text=Thus%2C%20the%20term%20frequency%20is,how%20important%20a%20term%20is.
   https://blog.skillfactory.ru/glossary/nlp/
10. https://uk.theastrologypage.com/tokenization