ОБРОБКА ПРИРОДНОЇ МОВИ (NLP) У PYTHON (КЛАСТЕРИЗАЦІЯ ТЕКСТІВ) ЛЕКЦІЯ 3

- Кластеризація тексту — це завдання угруповання набору текстів без міток таким чином, щоб тексти в одній групі (кластер) були більш схожі один на одного, ніж на тексти в інших кластерах.

- Багато алгоритмів кластеризації доступні в Scikit-Learn та інших бібліотеках, але, можливо, найпростішим для розуміння є алгоритм, відомий як кластеризація k-середніх, який реалізований у sklearn.cluster.KMeans

- k-середніх алгоритм пошуку для заздалегідь певної кількості кластерів в межах немічених багатовимірного масиву даних. Використовується проста концепція того, як виглядає оптимальна кластеризація:

- «Центр кластеру» — це середнє арифметичне всіх точок, що належать кластеру.
- Кожна точка знаходиться ближче до свого центру кластера, ніж до інших центрів кластера.
- Ці два припущення є основою моделі k-середніх.

- Нам потрібно отримати необроблений текст заголовків статей.

```
import pandas as pd
df =
pd.read_csv(‘https://raw.githubusercontent.com/olegdubetcky/Text-Classification-with-ML-Project/main/news.csv', encoding=’utf8')
```

- Виконаємо попередню обробку тексту для поліпшення роботи алгоритму. Тож, виконаємо очищення від стоп-слів та приведемо слова в нормальну форму. Це знижує рівень шуму, щоб алгоритм не надавав ваги, використовувати минулий час замість справжнього або щоб іменники у множині не зважали окремою частиною словника замість іменника в однині.

```
import requests
import nltk
nltk.download('stopwords')
nltk.download('punkt')
import string
from nltk.corpus import stopwords

url = 'https://raw.githubusercontent.com/olegdubetcky/Ukrainian-
Stopwords/main/ukrainian'
r = requests.get(url)
with open(nltk.data.path[0]+'/corpora/stopwords/ukrainian', 'wb') as
f:
 f.write(r.content)
# Retrieve HTTP meta-data
print(r.status_code)
print(r.headers['content-type'])
print(r.encoding)

import string
from nltk.corpus import stopwords
stopwords = stopwords.words("ukrainian")
```

СТВОРИМО ДВА СЛОВНИКОВИХ СПИСКА: ОДИН З ОСНОВАМИ І ОДИН ТІЛЬКИ З ТОКЕНІЗАЦІЄЙ.

```
totalvocab_normalized = []
totalvocab_tokenized = []
stop_words = frozenset(stopwords+list(string.punctuation))
for index, row in df.iterrows():
 sentences = nltk.sent_tokenize(row['title'].lower())
 for sentence in sentences:
 words = nltk.word_tokenize(sentence)
 without_stop_words = [word for word in words if not word in
stop_words]

 normal_words=[]
 for token in without_stop_words:
 p = morph.parse(token)[0]
 normal_words.append(p.normal_form)
 n_title = ' '.join(normal_words)
 totalvocab_tokenized.extend(without_stop_words)
 totalvocab_normalized.extend(normal_words)
 df.loc[index, 'normalized'] =n_title
```

- Зі словникових списків ми можемо зробити таблицю пошуку основ для повного слова. Однак, оскільки основи можуть ставитися до декількох слів, ця таблиця пошуку поверне тільки перше слово.

- Тепер можемо реалізувати алгоритм TF-IDF Vectorizer.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000,
 min_df=0.2, stop_words=stop_words,
 use_idf=True,tokenizer=None, ngram_range=(1,3))
vec_matrix = vectorizer.fit_transform(df['normalized'])
print(vec_matrix.shape)
(3293, 6013)
```

- В результаті виходить матриця розміром (3293, 6013) близько 6 000 слів.
- Тепер ми можемо приступити до кластеризації.
- Складно відповісти, скільки кластерів призначити даним. За допомогою KMeans дані занадто сильно згруповані разом для ієрархічної кластеризації або будь-якого алгоритму, який самостійно знаходить кластери. Тож оберемо метод оцінки кількості кластерів, які з'являться у наборі даних.

```
from sklearn.cluster import KMeans
import math
n_clusters = int(math.sqrt(df.shape[0] / 2) * 1.5)

km = KMeans(n_clusters=n_clusters)
km.fit(vec_matrix)
```

Тепер можна зв’язати кожну статтю з відповідним кластером.

```
clusters = km.labels_.tolist()

df[‘cluster’] = clusters
df.head()
```

Нарешті, можемо роздрукувати кластери і назви пов'язаних з ними статей.

```
print("Топ слова у кластері:")
print()

#сортувати центри кластера за близькістю до центроїда
order_centroids = km.cluster_centers_.argsort()[:, ::-1]

for i in range(n_clusters):
 print("Кластер %d слова:" % i, end='')

 for ind in order_centroids[i, :6]:
 print(' %s' % terms[ind], end=',')

 print()
 print()

 print("Кластер %d назви:" % i, end='')
 print()
 for title in df[df['cluster'] == i]['title'].values.tolist():
 print(' – %s' % title)
 print()
 print()

print()
print()
```

Як бачите, деякі з них краще за інших, але, грунтуючись виключно на назвах, K-Means, схоже, розділили статті на відносно послідовні кластери.