Лабораторна робота №4 Варіант № 6

Тема: Наївний машинний переклад та локально-сенситивне хешування

```
import nltk
import numpy as np
import requests
import pandas as pd
import io
import gzip
import random
import time
import string
import re
from collections import defaultdict
from nltk.corpus import twitter_samples, stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
```

ГОЛОВНИЙ ПЕРЕМИКАЧ: ДЕМОНСТРАЦІЙНИЙ РЕЖИМ

```
# True = Використовувати маленькі, випадкові дані (запускається за 30 сек)
# False = Спроба завантажити гігабайтні файли (вимагає >16GB RAM та >20GB місця)
DEMO_MODE = True

# Розмірність векторів (стандарт для fastText/MUSE)
EMBEDDING_DIM = 300
```

ДОПОМІЖНІ ФУНКЦІЇ (Завантаження даних)

```
def load_embeddings_gensim(url: str, max_words=50000) -> dict:
    """
    Завантажує та парсить вектори fastText .vec.gz.
    Використовує 'gensim' для сумісності з вашим кодом.
    """
    print(f"Завантаження векторів з {url} (це може зайняти > 30 хв)...")
    from gensim.models import KeyedVectors

    try:
        # Завантажуємо файл
        response = requests.get(url, stream=True)
        response.raise_for_status()

        # Створюємо тимчасовий файл для gensim
        temp_filename = 'temp_vectors.vec.gz'
        with open(temp_filename, 'wb') as f:
            f.write(response.content)

        print("Завантаження завершено. Завантаження в Gensim...")

        # Завантажуємо в gensim
        # Ми не можемо використати 'limit', оскільки 'index_to_key' буде неповним.
        # Завантажуємо повну модель. Це вимагає багато RAM.
        model = KeyedVectors.load_word2vec_format(temp_filename, binary=False)

        os.remove(temp_filename) # Видаляємо тимчасовий файл

        print(f"Успішно завантажено {len(model.index_to_key)} векторів.")
        return model

    except requests.exceptions.RequestException as e:
        print(f"ПОМИЛКА: Не вдалося завантажити файл: {e}")
        return None
    except Exception as e:
        print(f"ПОМИЛКА при обробці Gensim: {e}")
        return None
```

```
def get_dict(file_content_str: str) -> dict:
    """
    Парсить словник перекладів з текстового рядка.
    (Ваш код використовував pd.read_csv, але це уникає залежності від Pandas)
    """
    etof = {}
    for line in file_content_str.splitlines():
        parts = line.strip().split()
```

```
            if len(parts) == 2:
                etof[parts[0]] = parts[1]
        return etof
```

```
def load_dictionary(url: str) -> dict:
    """Завантажує тренувальний/тестовий словник."""
    print(f"Завантаження словника з {url}...")
    try:
        response = requests.get(url)
        response.raise_for_status()
        dictionary = get_dict(response.text)
        print(f"Успішно завантажено {len(dictionary)} пар слів.")
        return dictionary
    except requests.exceptions.RequestException as e:
        print(f"ПОМИЛКА: Не вдалося завантажити словник: {e}")
        return {}
```

```
def create_demo_data(vocab_size=500, dict_size=100, dim=EMBEDDING_DIM):
    """
    Створює іграшкові дані для DEMO_MODE.
    Вектори пов'язані (en = fr + noise), твіти використовують слова зі словника.
    """
    print("[DEMO_MODE] Створення випадкових, але логічних даних...")
    # Gensim KeyedVectors - це складний об'єкт.
    # В демо-режимі ми будемо використовувати прості словники Python {word: np.array}
    # Це вимагатиме невеликих змін у коді

    emb_fr = {}
    emb_en = {}
    vocab_fr = []
    vocab_en = []

    for i in range(vocab_size):
        word_fr = f"frword{i}"
        word_en = f"enword{i}"

        base_vector = np.random.rand(dim)
        emb_fr[word_fr] = base_vector

        noise = (np.random.rand(dim) - 0.5) * 0.1 # Маленьке відхилення
        emb_en[word_en] = base_vector + noise

        vocab_fr.append(word_fr)
        vocab_en.append(word_en)

    # 2. Словники перекладів (dict)
    train_dict = {}
    test_dict = {}
    for i in range(dict_size):
        train_dict[vocab_fr[i]] = vocab_en[i]
    for i in range(dict_size, dict_size + 50):
        if i < vocab_size:
            test_dict[vocab_fr[i]] = vocab_en[i]

    # 3. Фальшиві твіти
    print("[DEMO_MODE] Створення фальшивих твітів...")
    tweets = []
    for _ in range(100):
        random_words = random.sample(vocab_fr, 5) # Твіти мовою-1 (Французька)
        tweets.append(" ".join(random_words))

    # Додамо два "дуже схожих" твіти, щоб LSH міг їх знайти
    tweets[0] = f"{vocab_fr[1]} {vocab_fr[2]} {vocab_fr[3]} {vocab_fr[4]}"
    tweets[1] = f"{vocab_fr[1]} {vocab_fr[2]} {vocab_fr[3]} {vocab_fr[5]}" # Схожий

    print("[DEMO_MODE] Іграшкові дані створено.")
    return emb_fr, emb_en, train_dict, test_dict, tweets
```

ЗАВДАННЯ 3: Побудова матриць X та Y

```
def get_matrices(l1_l2, l1_embeddings, l2_embeddings):
    """
    (Функція з опису, адаптована для роботи з dict та gensim)
    """
    X_l = list()
    Y_l = list()

    # Перевіряємо, чи є 'l1_embeddings' об'єктом gensim чи простим dict
    is_gensim_l1 = hasattr(l1_embeddings, 'index_to_key')
```

```
    is_gensim_l2 = hasattr(l2_embeddings, 'index_to_key')

    l1_set = set(l1_embeddings.index_to_key) if is_gensim_l1 else set(l1_embeddings.keys())
    l2_set = set(l2_embeddings.index_to_key) if is_gensim_l2 else set(l2_embeddings.keys())

    for l1_word, l2_word in l1_l2.items():
        if l1_word in l1_set and l2_word in l2_set:
            X_l.append(l1_embeddings[l1_word])
            Y_l.append(l2_embeddings[l2_word])

    X = np.asarray(X_l)
    Y = np.asarray(Y_l)
    return X, Y
```

ЗАВДАННЯ 4: Пошук матриці R (Градієнтний спуск)

```
def compute_loss(X, Y, R):
    """(Функція з опису)"""
    m = X.shape[0]
    diff = np.dot(X, R) - Y
    loss = np.sum(diff**2) / m
    return loss

def compute_gradient(X, Y, R):
    """(Функція з опису)"""
    m = X.shape[0]
    gradient = np.dot(X.T, np.dot(X, R) - Y) * (2 / m)
    return gradient

def align_embeddings(X, Y, train_steps=100, learning_rate=0.0003, verbose=True):
    """(Функція з опису)"""
    print("Початок пошуку матриці R (Градієнтний спуск)...")
    np.random.seed(129)
    m, n = X.shape
    R = np.random.rand(n, n)

    for i in range(train_steps):
        if verbose and i % 10 == 0:
            print(f"Loss at iteration {i} is: {compute_loss(X, Y, R):.4f}")
        gradient = compute_gradient(X, Y, R)
        R -= learning_rate * gradient

    print(f"Loss at iteration {train_steps} is: {compute_loss(X, Y, R):.4f}")
    print("Пошук R завершено.")
    return R
```

ЗАВДАННЯ 5 & 6: Переклад та Оцінка (Оптимізовано)

```
def optimized_cosine_similarity(v1, v2_matrix_normalized):
    """
    Швидко обчислює косинусну подібність v1 до КОЖНОГО рядка
    в ПОПЕРЕДНЬО НОРМАЛІЗОВАНІЙ матриці v2.
    """
    v1_norm = v1 / (np.linalg.norm(v1) + 1e-8)
    # v2_matrix_normalized вже нормалізована
    similarities = np.dot(v2_matrix_normalized, v1_norm)
    return similarities

def translate(word, R, l1_embeddings, l2_emb_matrix_norm, l2_vocab_list):
    """
    (Оптимізована версія Завдання 5)
    Перекладає слово, використовуючи швидкий пошук.
    """
    # Перевірка на gensim/dict
    if hasattr(l1_embeddings, 'key_to_index'):
        if word not in l1_embeddings:
            return None
    else:
        if word not in l1_embeddings:
            return None

    l1_emb = l1_embeddings[word]
    l2_emb_pred = np.dot(l1_emb, R)

    distances = optimized_cosine_similarity(l2_emb_pred, l2_emb_matrix_norm)

    top_k_idx = np.argsort(distances)[-1:] # Беремо 1 найкращий
    return l2_vocab_list[top_k_idx[0]]
```

```
def test_translation_accuracy(l1_l2_test, R, l1_embeddings, l2_embeddings):
    """
    (Виконує Завдання 6)
    (Виправлена та оптимізована версія 'test_vocabulary' з опису)
    """
    print("\nОцінка точності перекладу на тестовій вибірці...")

    # 1. Готуємо дані мови-2 для швидкого пошуку
    if hasattr(l2_embeddings, 'index_to_key'): # Gensim
        l2_vocab_list = l2_embeddings.index_to_key
        l2_matrix = l2_embeddings.vectors
    else: # Dict
        l2_vocab_list = list(l2_embeddings.keys())
        l2_matrix = np.array(list(l2_embeddings.values()))

    # 2. Попередньо нормалізуємо матрицю мови-2
    l2_matrix_norms = np.linalg.norm(l2_matrix, axis=1)
    l2_matrix_normalized = l2_matrix / (l2_matrix_norms[:, np.newaxis] + 1e-8)

    correct = 0
    total = 0

    for l1_word, l2_true in l1_l2_test.items():
        # Перевіряємо, що обидва слова є в наших словниках
        l1_exists = (hasattr(l1_embeddings, 'key_to_index') and l1_word in l1_embeddings) or (l1_word in l1_embeddings)
        l2_exists = l2_true in l2_vocab_list

        if l1_exists and l2_exists:
            total += 1
            pred_l2 = translate(l1_word, R, l1_embeddings, l2_matrix_normalized, l2_vocab_list)

            if pred_l2 == l2_true:
                correct += 1

    if total == 0:
        return 0.0
    return correct / total
```

ЗАВДАННЯ 7 & 8: Векторизація твітів

```
# (Використовуємо функції з Лаби 2, але адаптуємо 'get_tweet_embedding')

def process_tweet_for_lsh(tweet, is_demo=False):
    """
    Спрощена обробка твітів для LSH.
    Якщо is_demo=True, використовує .split()
    """
    if is_demo:
        # Демо-твіти - це просто слова, розділені пробілом
        return tweet.split(' ')
    else:
        # Для реальних твітів
        stemmer = PorterStemmer()
        stopwords_english = stopwords.words('english')
        tweet = re.sub(r'\$\w*', '', tweet)
        tweet = re.sub(r'^RT[\s]+', '', tweet)
        tweet = re.sub(r'https?://[^\s\n\r]+', '', tweet)
        tweet = re.sub(r'#', '', tweet)
        tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
        tweet_tokens = tokenizer.tokenize(tweet)

        tweets_clean = []
        for word in tweet_tokens:
            if (word not in stopwords_english and word not in string.punctuation):
                # НЕ робимо стемінг, щоб знайти слова у векторах fastText
                tweets_clean.append(word)
        return tweets_clean

def get_tweet_embedding(tweet, l1_embeddings, is_demo=False):
    """(Виконує Завдання 8) - усереднення векторів"""
    words = process_tweet_for_lsh(tweet, is_demo)
    embeddings = []

    is_gensim = hasattr(l1_embeddings, 'key_to_index')

    for w in words:
        if (is_gensim and w in l1_embeddings) or (not is_gensim and w in l1_embeddings):
            embeddings.append(l1_embeddings[w])

    if not embeddings:
        return np.zeros(EMBEDDING_DIM)
```

```
    tweet_embedding = np.mean(embeddings, axis=0)
    return tweet_embedding

def get_all_tweet_embeddings(tweets, l1_embeddings, is_demo=False):
    """(Виконує Завдання 8 - для всіх твітів)"""
    tweet_embeddings = {}
    for i, tweet in enumerate(tweets):
        tweet_embeddings[i] = get_tweet_embedding(tweet, l1_embeddings, is_demo)
    return tweet_embeddings
```

ЗАВДАННЯ 9 & 10: Локально-Сенситивне Хешування (LSH)

```
def hash_func(embedding, planes):
    """(Виконує частину Завдання 9)"""
    hash_value = 0

    # *** ВИПРАВЛЕННЯ: ***
    # Ми повинні ітерувати по стовпцях матриці 'planes', а не по рядках.
    # 'planes' має розмір (300, 10). 'planes.T' (транспонована) має розмір (10, 300).
    # Ітерація по 'planes.T' дає нам 10 векторів (площин), кожен розміром (300,).
    for i, plane in enumerate(planes.T):
        # Тепер 'embedding' (300,) і 'plane' (300,) мають однакову розмірність
        sign = 1 if np.dot(embedding, plane) > 0 else 0
        hash_value += sign * 2 ** i
    return hash_value

def make_hash_table(embeddings, planes):
    """(Виконує частину Завдання 9)"""
    hash_table = defaultdict(list)
    for i, embedding in embeddings.items():
        # Не хешуємо нульові вектори
        if np.linalg.norm(embedding) > 0:
            hash_value = hash_func(embedding, planes)
            hash_table[hash_value].append(i)
    return hash_table

def init_lsh(embeddings_dict, n_planes, n_tables):
    """(Виконує Завдання 9: створює LSH таблиці)"""
    print(f"Створення LSH (planes={n_planes}, tables={n_tables})...")

    # Словник {id: vector} -> матриця (m, n)
    embeddings_matrix = np.array(list(embeddings_dict.values()))

    planes_list = [np.random.normal(size=(EMBEDDING_DIM, n_planes))
                   for _ in range(n_tables)]
    tables = [make_hash_table(embeddings_dict, planes) for planes in planes_list]
    print(f"Створено {len(tables)} LSH таблиць.")
    return tables, planes_list

def lsh_knn(tweet_id, embedding, tweet_embeddings_dict, tables, planes_list, k=5):
    """
    (Виконує Завдання 10: Пошук LSH)
    tweet_embeddings_dict: словник {id: vector}
    """
    candidates = set()
    for table, planes in zip(tables, planes_list):
        hash_value = hash_func(embedding, planes)
        candidates.update(table.get(hash_value, []))

    # Видаляємо сам запит зі списку кандидатів
    if tweet_id in candidates:
        candidates.discard(tweet_id)

    if not candidates:
        return []

    # 4. Обчислюємо реальну подібність ТІЛЬКИ для кандидатів
    candidate_indices = list(candidates)
    candidate_embeddings = np.array([tweet_embeddings_dict[i] for i in candidate_indices])

    # Використовуємо оптимізовану ф-цію
    similarities = optimized_cosine_similarity(embedding, candidate_embeddings)

    # Сортуємо
    top_k_local_idx = np.argsort(similarities)[-k:][::-1]

    # Повертаємо глобальні індекси
    neighbors = [(candidate_indices[i], similarities[i]) for i in top_k_local_idx]

    return neighbors
```

ГОЛОВНИЙ СКРИПТ (ЗАПУСК)

```
def main():
    print("--- Лабораторна робота № 4: Початок ---")
    print(f"Варіант 6: Французька (мова 1) -> Англійська (мова 2)")

    # --- URL-адреси (Французька -> Англійська) ---
    URL_EMB_FR = "https://dl.fbaipublicfiles.com/muse/vectors-in-fasttext-format/wiki.multi.fr.vec.gz"
    URL_EMB_EN = "https://dl.fbaipublicfiles.com/muse/vectors-in-fasttext-format/wiki.multi.en.vec.gz"
    URL_TRAIN_DICT = "https://dl.fbaipublicfiles.com/arrival/dictionaries/fr-en.0-5000.txt"
    URL_TEST_DICT = "https://dl.fbaipublicfiles.com/arrival/dictionaries/fr-en.5000-6500.txt"

    if DEMO_MODE:
        # (Завдання 1, 2, 7) - Генеруємо демо-дані
        lang1_embeddings, lang2_embeddings, l1_l2_train, l1_l2_test, all_tweets = create_demo_data()
    else:
        # (Завдання 1, 2, 7) - Завантажуємо реальні дані
        # УВАГА: Це вимагає >16GB RAM та >15GB місця
        lang1_embeddings = load_embeddings_gensim(URL_EMB_FR) # Французька
        lang2_embeddings = load_embeddings_gensim(URL_EMB_EN) # Англійська
        if lang1_embeddings is None or lang2_embeddings is None:
            print("Не вдалося завантажити вектори. Зупинка.")
            return

        l1_l2_train = load_dictionary(URL_TRAIN_DICT)
        l1_l2_test = load_dictionary(URL_TEST_DICT)

        nltk.download('twitter_samples', quiet=True)
        all_tweets = twitter_samples.strings('positive_tweets.json') + twitter_samples.strings('negative_tweets.json')

    # --- Частина 1: Машинний переклад ---
    print("\n--- Частина 1: Машинний переклад ---")

    # (Завдання 3)
    print("Побудова матриць X та Y...")
    X_train, Y_train = get_matrices(l1_l2_train, lang1_embeddings, lang2_embeddings)
    print(f"Розмір матриці X (train): {X_train.shape}")
    print(f"Розмір матриці Y (train): {Y_train.shape}")

    if X_train.shape[0] == 0:
        print("ПОМИЛКА: Не вдалося знайти спільних слів у словниках.")
        return

    # (Завдання 4)
    R_train = align_embeddings(X_train, Y_train, train_steps=400, learning_rate=0.0003)

    # (Завдання 6)
    accuracy = test_translation_accuracy(l1_l2_test, R_train, lang1_embeddings, lang2_embeddings)
    print(f"Точність перекладу (Accuracy @ 1): {accuracy * 100:.2f}%")

    # (Завдання 5 - демонстрація)
    print("\nДемонстрація перекладу (Завдання 5):")
    test_word_fr = list(l1_l2_train.keys())[0] # Беремо перше слово з train dict
    if not DEMO_MODE:
        test_word_fr = 'chien' # Собака

    # Готуємо дані для швидкого перекладу
    if hasattr(lang2_embeddings, 'index_to_key'): # Gensim
        l2_vocab_list = lang2_embeddings.index_to_key
        l2_matrix = lang2_embeddings.vectors
    else: # Dict
        l2_vocab_list = list(lang2_embeddings.keys())
        l2_matrix = np.array(list(lang2_embeddings.values()))
    l2_matrix_norms = np.linalg.norm(l2_matrix, axis=1)
    l2_matrix_normalized = l2_matrix / (l2_matrix_norms[:, np.newaxis] + 1e-8)

    translation = translate(test_word_fr, R_train, lang1_embeddings, l2_matrix_normalized, l2_vocab_list)
    print(f"Французьке: '{test_word_fr}' -> Англійське: '{translation}'")

    # --- Частина 2: Локально-Сенситивне Хешування (LSH) ---
    print("\n--- Частина 2: LSH для твітів ---")

    # (Завдання 8) - Векторизуємо твіти (використовуючи вектори FR)
    print(f"Векторизація {len(all_tweets)} твітів (використовуючи вектори FR)...")
    tweet_embeddings = get_all_tweet_embeddings(all_tweets, lang1_embeddings, is_demo=DEMO_MODE)

    # (Завдання 9)
    tables, planes_list = init_lsh(tweet_embeddings, n_planes=10, n_tables=5)

    # (Завдання 11 - Тестування LSH)
    query_id = 0 # Беремо перший твіт як запит
```

```
        query_id = 0 # Беремо перший твіт як запит
        query_tweet = all_tweets[query_id]
        query_embedding = tweet_embeddings[query_id]

        print(f"\n--- Тестування LSH (Завдання 11) ---")
        print(f"Твіт-запит (індекс {query_id}):\n{query_tweet}\n")
        print("Шукаю подібні твіти...")

        start_time = time.time()
        # (Завдання 10 - виклик функції пошуку)
        neighbors = lsh_knn(query_id, query_embedding, tweet_embeddings, tables, planes_list, k=3)
        end_time = time.time()

        print(f"Пошук LSH зайняв {end_time - start_time:.6f} секунд.")

        print("\nТоп-3 найбільш подібних твітів:")
        if not neighbors:
            print("Схожих твітів у тих самих 'відрах' не знайдено.")

        for neighbor_id, similarity in neighbors:
            print(f"  (Індекс: {neighbor_id}, Подібність: {similarity:.4f})")
            print(f"     {all_tweets[neighbor_id][:80]}...")

        print("\n--- Лабораторна робота № 4: Завершено ---")

    # Запускаємо головну функцію
    if __name__ == "__main__":
        main()
```

```
--- Лабораторна робота № 4: Початок ---
Варіант 6: Французька (мова 1) -> Англійська (мова 2)
[DEMO_MODE] Створення випадкових, але логічних даних...
[DEMO_MODE] Створення фальшивих твітів...
[DEMO_MODE] Іграшкові дані створено.

--- Частина 1: Машинний переклад ---
Побудова матриць X та Y...
Розмір матриці X (train): (100, 300)
Розмір матриці Y (train): (100, 300)
Початок пошуку матриці R (Градієнтний спуск)...
Loss at iteration 0 is: 1665001.0154
Loss at iteration 10 is: 662560.9103
Loss at iteration 20 is: 263890.6053
Loss at iteration 30 is: 105338.5462
Loss at iteration 40 is: 42281.1184
Loss at iteration 50 is: 17201.7529
Loss at iteration 60 is: 7226.2063
Loss at iteration 70 is: 3257.4303
Loss at iteration 80 is: 1677.5440
Loss at iteration 90 is: 1047.7223
Loss at iteration 100 is: 795.7466
Loss at iteration 110 is: 694.0467
Loss at iteration 120 is: 652.1182
Loss at iteration 130 is: 633.9675
Loss at iteration 140 is: 625.2802
Loss at iteration 150 is: 620.3632
Loss at iteration 160 is: 616.9526
Loss at iteration 170 is: 614.1478
Loss at iteration 180 is: 611.5906
Loss at iteration 190 is: 609.1387
Loss at iteration 200 is: 606.7353
Loss at iteration 210 is: 604.3578
Loss at iteration 220 is: 601.9973
Loss at iteration 230 is: 599.6500
Loss at iteration 240 is: 597.3146
Loss at iteration 250 is: 594.9903
Loss at iteration 260 is: 592.6770
Loss at iteration 270 is: 590.3745
Loss at iteration 280 is: 588.0827
Loss at iteration 290 is: 585.8015
Loss at iteration 300 is: 583.5308
Loss at iteration 310 is: 581.2707
Loss at iteration 320 is: 579.0210
Loss at iteration 330 is: 576.7817
Loss at iteration 340 is: 574.5527
Loss at iteration 350 is: 572.3341
Loss at iteration 360 is: 570.1256
Loss at iteration 370 is: 567.9274
Loss at iteration 380 is: 565.7393
Loss at iteration 390 is: 563.5612
Loss at iteration 400 is: 561.3932
Пошук R завершено.

Оцінка точності перекладу на тестовій вибірці...
Точність перекладу (Accuracy @ 1): 0.00%

Демонстрація перекладу (Завдання 5):
```