



# КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №2

## Визначення сорту вина за його описом

*Пядика Любомира, МІ-4*

30 листопада 2018 р.

# 1 Постановка задачі

Необхідно розв'язати задачу знаходження найімовірнішого сорту вина, яке описано експертом у словесному вигляді. При реалізації використати уніграми і нейронні мережі, які гратимуть ключову роль при визначенні сорту вина. Для тренування вибрати певну кількість найчастіших уніграм із усіх описів вин, орієнтуватись на їх входження у певний опис, векторизувати, і ці вектори слугуватимуть як вхід до нейронної мережі, а на виході один із можливих сортів вина.

## 2 Теоретичні відомості

*N-грама* [1] — послідовність з  $n$  елементів. З семантичної точки зору, це може бути послідовність звуків, складів, слів або букв. На практиці частіше зустрічається  $N$ -грами як ряд слів, стійкі словосполучення називають колокацією. Послідовність з двох послідовних елементів часто називають біграм, послідовність з трьох елементів називається триграма. Не менш чотирьох і вище елементів позначаються як  $N$ -грами,  $N$  замінюється на кількість послідовних елементів.

*Штучні нейронні мережі* [4] (*ШНМ*, англ. *artificial neural networks, ANN*) — це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як “кіт” і “не кіт”, і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють.

*Глибинне навчання* [3] (також відоме як глибинне структурне навчання, ієрархічне навчання, глибинне машинне навчання, англ. *deep learning, deep structured learning, hierarchical learning, deep machine learning*) — це галузь машинного навчання, що ґрунтується на наборі алгоритмів, які намагаються моделювати високорівневі абстракції в даних, застосовуючи глибинний граф із декількома обробними шарами, що побудовано з кількох лінійних або нелінійних перетворень.

Глибинне навчання є частиною ширшого сімейства методів машинного навчання, що ґрунтуються на навчанні ознак даних. Спостереження (наприклад, зображення) може бути представлено багатьма способами, такими як вектор значень яскравості для пікселів, або абстрактнішим способом, як множина кромek, областей певної форми тощо. Деякі представлення є кращими за інші у спрощенні задачі навчання (наприклад, розпізнаванню облич, або виразів облич). Однією з обіцянок глибинного навчання є заміна ознак ручної роботи дієвими алгоритмами автоматичного або напіваавтоматичного навчання ознак та ієрархічного виділення ознак.

## 3 Програмна реалізація

Лабораторну роботу було написано на мові програмування Python. Програма складається із 2 файлів (`main.py` - головний файл, який читає датасет, виконує навчання, і виводить одержану точність на тестовій частині датасету; `get_top_xwords.py` - реалізовує допоміжні функції, які займаються знаходженням найчастіших уніграм у датасеті та векторизацією описів).

По-перше, нам доведеться реструктурувати дані таким чином, щоб його легко було обробляти та зрозуміти наша нейронна мережа. Ми можемо це зробити, замінивши слова однозначно ідентифікуючими числами. Представляючи слова таким чином, ми отримуємо гнучким і семантично чутливий підхід.

Є сенс зосередитися на часто використовуваних словах, а також на фільтрації найбільш часто використовуваних слів (наприклад, “.”, “,”, “the”, “a”).

Ми можемо реалізувати цю функцію за допомогою Defaultdict та NLTK, який розміщений у `get_top_xwords.py`.

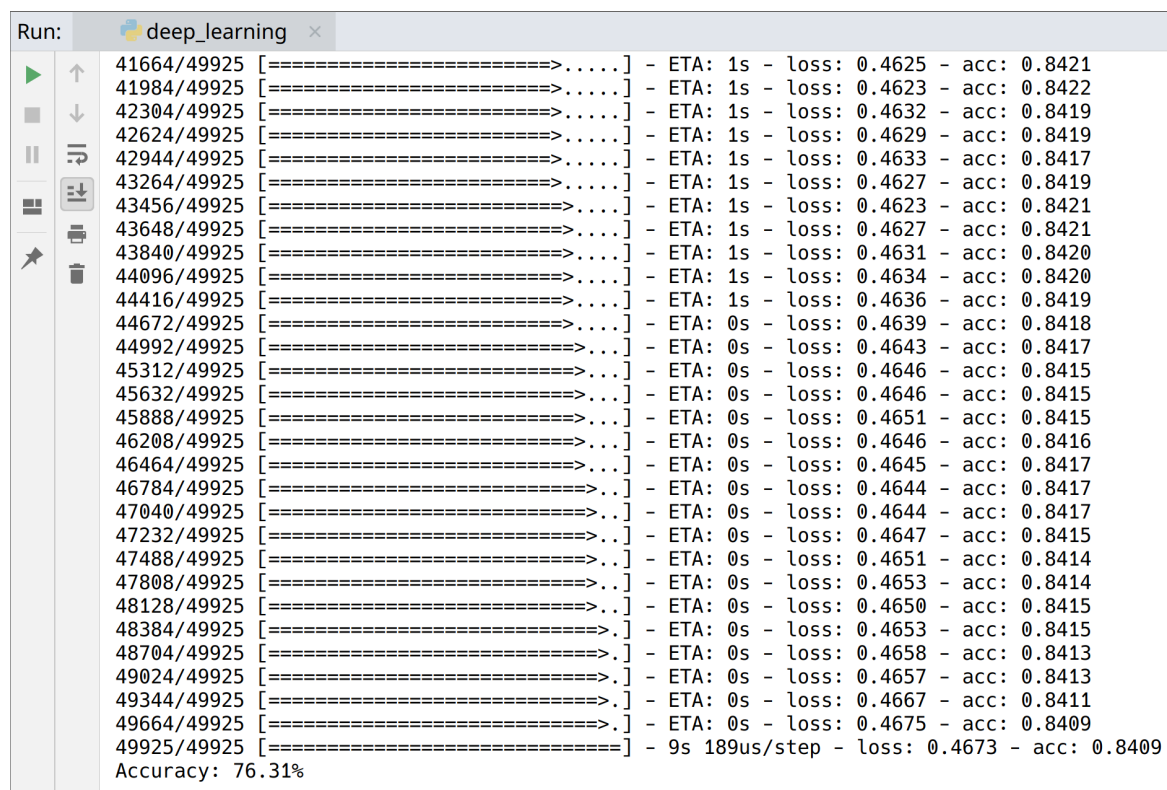
Ми будемо використовувати Keras з Tensorflow для побудови нашої моделі. Keras - це бібліотека Python, яка дозволяє створювати глибокі навчальні моделі дуже легко порівняно з низьким рівнем інтерфейсу API Tensorflow. На додаток до dense шарів, ми також будемо використовувати embedding та convolutional шари, щоб вивчити основну семантичну інформацію слів і потенційні структурні схеми в даних.

Для тестування було використано 30% із усього датасету, а решта 70% для тренування. Датасет знаходиться у публічному доступі на сайті Kaggle [2].

Для використання необхідно встановити наступні пакети у Python:

```
pip install nltk
pip install numpy
pip install pandas
pip install sklearn
pip install keras
pip install tensorflow
```

### 3.1 Приклад виконання програми



```
Run: deep_learning x
41664/49925 [=====>.....] - ETA: 1s - loss: 0.4625 - acc: 0.8421
41984/49925 [=====>.....] - ETA: 1s - loss: 0.4623 - acc: 0.8422
42304/49925 [=====>.....] - ETA: 1s - loss: 0.4632 - acc: 0.8419
42624/49925 [=====>.....] - ETA: 1s - loss: 0.4629 - acc: 0.8419
42944/49925 [=====>.....] - ETA: 1s - loss: 0.4633 - acc: 0.8417
43264/49925 [=====>.....] - ETA: 1s - loss: 0.4627 - acc: 0.8419
43456/49925 [=====>.....] - ETA: 1s - loss: 0.4623 - acc: 0.8421
43648/49925 [=====>.....] - ETA: 1s - loss: 0.4627 - acc: 0.8421
43840/49925 [=====>.....] - ETA: 1s - loss: 0.4631 - acc: 0.8420
44096/49925 [=====>.....] - ETA: 1s - loss: 0.4634 - acc: 0.8420
44416/49925 [=====>.....] - ETA: 1s - loss: 0.4636 - acc: 0.8419
44672/49925 [=====>.....] - ETA: 0s - loss: 0.4639 - acc: 0.8418
44992/49925 [=====>.....] - ETA: 0s - loss: 0.4643 - acc: 0.8417
45312/49925 [=====>.....] - ETA: 0s - loss: 0.4646 - acc: 0.8415
45632/49925 [=====>.....] - ETA: 0s - loss: 0.4646 - acc: 0.8415
45888/49925 [=====>.....] - ETA: 0s - loss: 0.4651 - acc: 0.8415
46208/49925 [=====>.....] - ETA: 0s - loss: 0.4646 - acc: 0.8416
46464/49925 [=====>.....] - ETA: 0s - loss: 0.4645 - acc: 0.8417
46784/49925 [=====>.....] - ETA: 0s - loss: 0.4644 - acc: 0.8417
47040/49925 [=====>.....] - ETA: 0s - loss: 0.4644 - acc: 0.8417
47232/49925 [=====>.....] - ETA: 0s - loss: 0.4647 - acc: 0.8415
47488/49925 [=====>.....] - ETA: 0s - loss: 0.4651 - acc: 0.8414
47808/49925 [=====>.....] - ETA: 0s - loss: 0.4653 - acc: 0.8414
48128/49925 [=====>.....] - ETA: 0s - loss: 0.4650 - acc: 0.8415
48384/49925 [=====>.....] - ETA: 0s - loss: 0.4653 - acc: 0.8415
48704/49925 [=====>.....] - ETA: 0s - loss: 0.4658 - acc: 0.8413
49024/49925 [=====>.....] - ETA: 0s - loss: 0.4657 - acc: 0.8413
49344/49925 [=====>.....] - ETA: 0s - loss: 0.4667 - acc: 0.8411
49664/49925 [=====>.....] - ETA: 0s - loss: 0.4675 - acc: 0.8409
49925/49925 [=====] - 9s 189us/step - loss: 0.4673 - acc: 0.8409
Accuracy: 76.31%
```

Рис. 1: Запущений main.py файл у PyCharm

## 3.2 Лістинг програми

main.py

```
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.utils import to_categorical
import pandas as pd
from collections import Counter
from sklearn.model_selection import train_test_split
from text_classification.get_top_xwords import filter_to_top_x

df = pd.read_csv('../data/wine_data.csv')

counter = Counter(df['variety'].tolist())
varieties_cnt = 10
top_varieties = {i[0]: idx for idx, i in enumerate(counter.most_common(varieties_cnt))}
print(top_varieties)
df = df[df['variety'].map(lambda x: x in top_varieties)]

description_list = df['description'].tolist()
# print(description_list)
mapped_list, word_list = filter_to_top_x(description_list, 2500, 10)
varietal_list_o = [top_varieties[i] for i in df['variety'].tolist()]
varietal_list = to_categorical(varietal_list_o)

max_length = 150
mapped_list = sequence.pad_sequences(mapped_list, maxlen=max_length)
train_x, test_x, train_y, test_y = train_test_split(mapped_list, varietal_list, test_size=0.2)

embedding_vector_length = 64
model = Sequential()
model.add(Embedding(2500, embedding_vector_length, input_length=max_length))
model.add(Conv1D(25, 5))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(varieties_cnt, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(train_x, train_y, epochs=4, batch_size=64)

y_score = model.predict(test_x)
y_score = [[1 if i == max(sc) else 0 for i in sc] for sc in y_score]
n_right = 0
for i in range(len(y_score)):
    if all(y_score[i][j] == test_y[i][j] for j in range(len(y_score[i]))):
        n_right += 1

print("Accuracy: %.2f%%" % (n_right / len(test_y) * 100))

get_top_xwords.py
```

```

import nltk
nltk.download('punkt')
from nltk import word_tokenize
from collections import defaultdict

def count_top_words(corpus, top_x, skip_top_n):
    count = defaultdict(lambda: 0)
    for c in corpus:
        for w in word_tokenize(c):
            count[w] += 1
    count_tuples = sorted([(w, c) for w, c in count.items()], key=lambda x: x[1], reverse=True)
    return [i[0] for i in count_tuples[skip_top_n: skip_top_n + top_x]]

def replace_top_words_with_vectors(corpus, top_x):
    top_x_dict = {top_x[i]: i for i in range(len(top_x))}

    return [
        [top_x_dict[w] for w in word_tokenize(s) if w in top_x_dict]
        for s in corpus
    ], top_x_dict

def filter_to_top_x(corpus, n_top, skip_n_top=0):
    top_x = count_top_words(corpus, n_top, skip_n_top)
    return replace_top_words_with_vectors(corpus, top_x)

```

## Література

- [1] N-gram. <https://en.wikipedia.org/wiki/N-gram>.
- [2] Wine reviews. <https://www.kaggle.com/zynicide/wine-reviews>.
- [3] Глибинне навчання. [https://uk.wikipedia.org/wiki/Глибинне\\_навчання](https://uk.wikipedia.org/wiki/Глибинне_навчання).
- [4] Штучна нейронна мережа. [https://uk.wikipedia.org/wiki/Штучна\\_нейронна\\_мережа](https://uk.wikipedia.org/wiki/Штучна_нейронна_мережа).