Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №1

з дисципліни «Системи штучного інтелекту»

на тему

«Моделі машинного навчання»

Виконав:

студент гр. ІС-91

Косюк Михайло

Київ – 2021

**Датасет**

У лабораторній роботі використовується стандартний датасет Titanic, що описує виживаємість пасажирів Титаніка. Датасет має наступні дані про пасажирів:

'age': tf.float32 – **Вік пасажира**

'sex': ClassLabel(shape=(), dtype=tf.int64, num\_classes=2) – **Стать пасажира**

    'sibsp': tf.int32,- **Кількість родичів на борту**

    'parch': tf.int32, - **Кількість дітей на борту**

    'fare': tf.float32, - **Тариф пасажира**

    'pclass': ClassLabel(shape=(), dtype=tf.int64, num\_classes=3), - **Клас білета**  
    'deck': tf.string, - **Палуба**  
    'embarked': ClassLabel(shape=(), dtype=tf.int64, num\_classes=4), - **Місце посадки**  
    'alone': tf.string, - **Самотній**

'survived': ClassLabel(shape=(), dtype=tf.int64, num\_classes=2), - **Чи вижив пасажир**

Всі дані, окрім survived використовуються як вхідні дані, survived є єдиними вихідними даними.

**Задача**

Задачею роботи є задача класифікації на два класи, зокрема для вибраного датасету – класифікація пасажирів за виживаємістю, Постановка задачі - визначити вижив пасажир (1 клас - **1**), чи загинув (2 клас - **0**). Вихідні дані - клас survived.

**Config**

TRAIN\_DATA\_URL = "https://storage.googleapis.com/tf-datasets/titanic/train.csv"

TEST\_DATA\_URL = "https://storage.googleapis.com/tf-datasets/titanic/eval.csv"

BATCH\_SIZE = 30

TRAINING\_EPOCHS = 20

SHUFFLE\_TIMES = 500

LABEL\_COLUMN = 'survived'

LABELS = [0, 1]

**Підготовка даних**

1. Завантаження датасетів tensorflow, навчальної та тестової вибірок:

# Get dataset files

train\_file\_path = tf.keras.utils.get\_file("train.csv", TRAIN\_DATA\_URL)

test\_file\_path = tf.keras.utils.get\_file("eval.csv", TEST\_DATA\_URL)

def get\_dataset(file\_path):

  dataset = tf.data.experimental.make\_csv\_dataset(

      file\_path,

      batch\_size=BATCH\_SIZE,

      label\_name=LABEL\_COLUMN,

      na\_value="?",

      num\_epochs=1,

      ignore\_errors=True)

  return dataset

raw\_train\_data = get\_dataset(train\_file\_path)

raw\_test\_data = get\_dataset(test\_file\_path)

1. Конвертація даних до типів tensorflow. Створення categorical колонок

CATEGORIES = {

    'sex': ['male', 'female'],

    'class' : ['First', 'Second', 'Third'],

    'deck' : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],

    'embark\_town' : ['Cherbourg', 'Southhampton', 'Queenstown'],

    'alone' : ['y', 'n']

}

categorical\_columns = []

for feature, vocab in CATEGORIES.items():

  cat\_col = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

        key=feature, vocabulary\_list=vocab)

  categorical\_columns.append(tf.feature\_column.indicator\_column(cat\_col))

1. Конвертація даних до типів tensorflow. Створення numeric колонок. Середні значення колонок

MEANS = {

    'age' : 29.631308,

    'n\_siblings\_spouses' : 0.545455,

    'parch' : 0.379585,

    'fare' : 34.385399

}

1. Конвертація даних до типів tensorflow. Створення numeric колонок. Функція нормалізації та створення

# Create numerical columns by normalizing number data

def process\_continuous\_data(mean, data):

  # Normalize data

  data = tf.cast(data, tf.float32) \* 1/(2\*mean)

  return tf.reshape(data, [-1, 1])

numerical\_columns = []

for feature in MEANS.keys()

  normalizer\_fn = lambda data, mean=MEANS[feature]: process\_continuous\_data(mean,data)

  num\_col = tf.feature\_column.numeric\_column(feature, normalizer\_fn=normalizer\_fn)

  numerical\_columns.append(num\_col)

1. Створення вхідного шару на основі створених tensorflow колонок

preprocessing\_layer = tf.keras.layers.DenseFeatures(categorical\_columns+numerical\_columns)

**Опис моделі**

Для вирішення задачі використовувалася модель машинного навчання багатошаровий персептрон. Персептрон складається із 5 шарів:

* Preprocessing layer
* 3 hidden layers (2 with relu activation and 1 with tanh activation function)
* Output layer with sigmoid activation function

Перший шар містить 9 вхідних нейронів, останній шар містить один вихідний нейрон із функцією активації sigmoid(x) = 1 / (1 + exp(-x)), що повертає значення від 0 до 1, таким чином визначаючи ближче до якого класу survived відноситься пасажир.

Для навчання використовується форма навчання “із вчителем”, тобто у тренувальному датасеті окрім вхідних значень присутнє і очікуване вихідне значення (survived).

model = tf.keras.Sequential([

  preprocessing\_layer,

  tf.keras.layers.Dense(128, activation='relu'),

  tf.keras.layers.Dense(128, activation='relu'),

  tf.keras.layers.Dense(128, activation='tanh'),

  tf.keras.layers.Dense(1, activation='sigmoid'),

])

**Навчання та тестування моделі**

Для оцінки втрат використовується функція binary\_crossentropy, оскільки ми маємо класи зі значеннями 0 и 1, яка показую наскільки отримані при навчанні результати відрізняються еталонних результатів.

model.compile(

    loss='binary\_crossentropy',

    optimizer='adam',

    metrics=['accuracy'])

Перед навчанням тренувальний датасет перемішується, щоб не повторювати послідовність при навчанні.

# Preparing shuffled data for training

train\_data = raw\_train\_data.shuffle(SHUFFLE\_TIMES)

test\_data = raw\_test\_data

Тренування із 20 епохами по 30 тензорів:

# Training

model.fit(train\_data, epochs=TRAINING\_EPOCHS)

Тестування натренованої моделі на тестовому датасеті, тестовий датасет у даному випадку в 3 рази менший за тренувальний.

#Evaluating

test\_loss, test\_accuracy = model.evaluate(test\_data)

print('\n\nTest Loss {}, Test Accuracy {}'.format(test\_loss, test\_accuracy))

**Оцінка результатів навчання**

Після навчання та тестування отримаємо наступні результати





Бачимо, що при навчанні використовуючи побудовану модель вдалося досягти точності в 87.56% із середніми втратами 31.97%. Що є не дуже високими результатами, однією з причин цього може бути невеликий розмір тренувального датасету у 600 тензорів, який може не покривати всіх або майже всіх можливих випадків. При наявності більшого датасета, з великою ймовірністю можна було б досягти кращої точності.

Також можна побачити, що при тестуванні значення точності є меншим, а функції втрат навпаки більшими. В даному випадку можливо, що тренувальний датасет не покрив усіх випадків тестового, але це є малоймовірним. Також в даному випадку може мати місце явище перенавчання.

**Перенавчання (overfitting)** - одна з проблем глибоких нейронних мереж (Deep Neural Networks, DNN), що складається в наступному: модель добре пояснює тільки приклади з навчальної вибірки, адаптуючись до навчальних прикладів, замість того щоб вчитися класифікувати приклади, які не брали участі в навчанні (втрачаючи здатність до узагальнення).

Найкращим методом боротьби із цим є метод проріжування (Dropout), який полягає в тому щоб викидати із мережі деякі нейрони на кожному шарі, таким чином утворюючи декілька варіацій нейронної мережі із одної, а потім знаходити їх усереднене значення.

