МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

МЕТОДИЧНІ ВКАЗІВКИ
ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни
Прикладні методи аналізу даних
(назва)

для PHD студентів

Розробник: проф., д. т. н. Новотарський М. А.
(посада, вчений ступінь та звання П.І.Б.)

Ухвалено
на засіданні кафедри ОТ
протокол № 15 від 29.05.2024 р.

Погоджено
на засіданні методичної
комісії ФІОТ
протокол №10 від 21.06.2024 р.

Київ – 2024

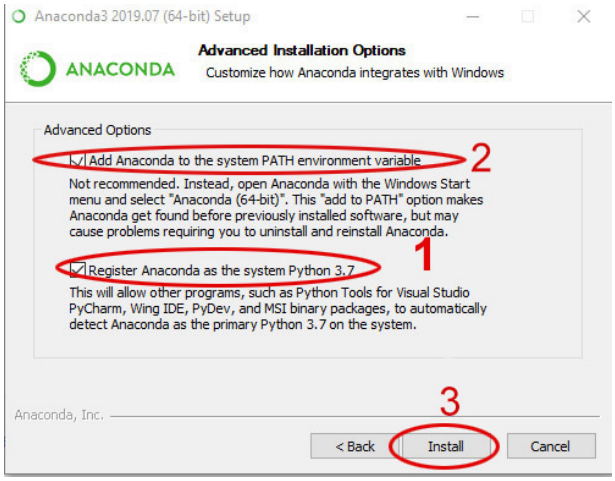
Метою проведення циклу лабораторних робіт є набуття студентами необхідних практичних навичок складних обчислень, які використовуються при обробці даних, використання методів машинного навчання для розв'язування задач класифікації, кластеризації і регресії (прогнозування), побудови деяких основних класифікаторів з використанням агрегатора Anaconda3 з Jupyter notebook, бібліотек `TensorFlow,Pandas, NumPy, Matplorlib.`

Зміст

Попередні інсталяції

1.Перед інсталяцією бібліотеки Theano необхідно спочатку з**авантажити дистрибутив** «Anaconda 3» **з сайту за посиланням** https://www.anaconda.com/distribution/

При інсталяції встановити таке налаштування:



3. Інсталювати TensorFlow

```
conda install tensorflow
```

4. Інсталювати PyTorch

```
conda install tensorflow
```

Лабораторна робота № 1

Тема. Попередня обробка даних

Мета. Навчитися виконувати попередню обробку даних з використанням бібліотек Scikit Learn та Pandas

Теоретичні відомості

Бібліотека Scilit Learn.

Scikit-learn (Sklearn) — це найпростіша та надійна бібліотека для машинного навчання на Python. Вона надає вибір ефективних інструментів для машинного навчання та статистичного моделювання, включаючи класифікацію, регресію, кластеризацію та зменшення розмірності через узгоджений інтерфейс на Python. Ця бібліотека, яка в основному написана на Python, побудована на NumPy, SciPy і Matplotlib.

Інсталяція бібліотеки з використанням `pip`

Наступну команду можна використовувати для встановлення scikit-learn через pip .

```
pip install -U scikit-learn
```

Інсталяція бібліотеки з використанням *`conda`*

```
conda install scikit-learn
```

З іншого боку, якщо NumPy і Scipy ще не встановлено на вашій робочій станції Python, ви можете встановити їх за допомогою pip або conda.

Іншим варіантом використання scikit-learn є використання дистрибутивів `Python` та `Anaconda`, оскільки вони обидва постачають останню версію scikit-learn.

Особливості

Деякі з найпопулярніших груп моделей, які використовує наданих Sklearn такі:

1. *Алгоритми навчання з учителем*

Майже всі популярні алгоритми навчання з учителем, як-от лінійна регресія, метод опорних векторів (SVM), дерево рішень тощо, є частиною scikit-learn.

2. *Алгоритми навчання без учителя*, такі як алгоритми кластеризації, факторного аналізу, PCA (аналіз основних компонентів) до нейронних мереж.

Моделі

Кластеризація – Ця модель використовується для групування немаркованих даних.

Перехресна перевірка – використовується для перевірки точності контрольованих моделей на невидимих даних.

Зменшення розмірності – використовується для зменшення кількості атрибутів у даних, які можна надалі використовувати для підсумовування, візуалізації та вибору ознак.

Методи ансамблю – як випливає з назви, він використовується для об'єднання прогнозів кількох контрольованих моделей.

Вилучення ознак – використовується для вилучення ознак із даних для визначення атрибутів у даних зображення та тексту.

Вибір функцій – використовується для визначення корисних атрибутів для створення контрольованих моделей.

Завантаження набору даних

Набір даних має такі два компоненти:

Ознаки – це змінні що описують об'єкти і називаються їх характеристиками. Їх також називають предикторами, входами або атрибутами.

Матриця ознак – це сукупність ознак, якщо їх більше однієї.

Назви ознак – це список усіх назв ознак.

Відповідь – це вихідна змінна, яка в основному залежить від змінних ознак. Вони також відомі як *target, label* або *output*.

Вектор відповіді – використовується для представлення стовпця відповіді. Зазвичай ми маємо лише одну колонку відповідей.

Цільові назви – представляють можливі значення, прийняті вектором відповіді.

Приклад 1.

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of X:\n", X[:10])
```

Результат

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal
length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
First 10 rows of X:
[
   [5.1 3.5 1.4 0.2]
   [4.9 3. 1.4 0.2]
   [4.7 3.2 1.3 0.2]
   [4.6 3.1 1.5 0.2]
   [5. 3.6 1.4 0.2]
   [5.4 3.9 1.7 0.4]
   [4.6 3.4 1.4 0.3]
   [5. 3.4 1.5 0.2]
   [4.4 2.9 1.4 0.2]
   [4.9 3.1 1.5 0.1]
]
```

Хід роботи

Загрузка даних

1. Завантажити датасет за посиланням:

Дані представлені як csv таблиці. https://www.kaggle.com/code/prasanshasatpathy/heart-failure-clinical-data-analysis

2.Створити Python скрипт. Завантажити датасет у датафрейм, та виключити бінарні ознаки та ознака часу.

```
import pandas as pd
import numpy as np
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
df = df.drop(columns =
['anaemia','diabetes','high_blood_pressure','sex','smoking','time'
,'DEATH_EVENT'])
print(df) #Вивід датафрейма з даними для лаб. роботи.
```

3.Побудова гістограм ознак

```
n_bins = 20
fig, axs = plt.subplots(2,3)
axs[0, 0].hist(df['age'].values, bins = n_bins)
axs[0, 0].set_title('age')
axs[0, 1].hist(df['creatinine_phosphokinase'].values, bins =
n_bins)
axs[0, 1].set_title('creatinine_phosphokinase')
axs[0, 2].hist(df['ejection_fraction'].values, bins = n_bins)
axs[0, 2].set_title('ejection_fraction')
axs[1, 0].hist(df['platelets'].values, bins = n_bins)
axs[1, 0].set_title('platelets')
axs[1, 1].hist(df['serum_creatinine'].values, bins = n_bins)
axs[1, 1].set_title('serum_creatinine')
axs[1, 2].hist(df['serum_sodium'].values, bins = n_bins)
axs[1, 2].set_title('serum_sodium')
plt.show()
```

4.Порівняйте дані до та після стандартизації. Опишіть, що змінилося і чому.

5.Розрахуйте мат. очікування та дисперсію до та після стандартизації. На підставі цих значень виведіть для кожної ознаки формули, за якими вони стандартизувалися.

6. Порівняйте значень формул з полями `mean_` і `var_` об'єкта `scaler`

7. Проведіть налаштування стандартизації на всіх даних та порівняйте з результатами налаштування на підставі 150 спостережень

Примітка: замість двох методів `fit` і `transform` можна використовувати метод `fit_transfrom`, щоб одразу налаштувати параметри та перетворити дані.

Приведення до діапазону

1.Приведіть дані до діапазону за допомогою https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler

2. Побудуйте гістограми для ознак та порівняйте з вихідними даними

3. Через параметри MinMaxScaler визначте мінімальне та максимальне значення даних для кожної ознаки

4. Аналогічно трансформуйте дані за допомогою https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler

та

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler

Побудуйте гістограми. Визначте, до якого діапазону наводяться дані.

5. Напишіть функцію, яка приводить усі дані до діапазону [-5 10]

Нелінійні перетворення

1. Приведіть дані до рівномірного розподілу за допомогою https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html#sklearn.preprocessing.QuantileTransformer

```
quantile_transformer =
preprocessing.QuantileTransformer(n_quantiles = 100,
random_state=0).fit(data)
data_quantile_scaled = quantile_transformer.transform(data)
```

2. Побудуйте гістограми та порівняйте з вихідними даними
3. Визначте, як і на що впливає значення параметра *n_quantiles*
4. Приведіть дані до нормального розподілу, передавши в *QuantileTransformer* параметр `output_distribution='normal'`
5. Побудуйте гістограми та порівняйте з вихідними даними
6. Самостійно приведіть дані до нормального розподілу за допомогою https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html#sklearn.preprocessing.PowerTransformer

Дискретизація ознак

1.Проведіть дискретизацію ознак, використовуючи https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html#sklearn.preprocessing.KBinsDiscretizer

наступну кількість діапазонів:

age - 3

creatinine_phosphokinase - 4

ejection_fraction - 3

platelets - 10

serum_creatinine - 2

serum_sodium - 4

2. Побудуйте гістограми. Поясніть отримані результати

3. Через параметр *bin_edges_* виведіть діапазони кожного інтервалу для кожної ознаки

ЗАВДАННЯ

1. Вибрати набір даних, користуючись порталом https://www.kaggle.com/
2. Виконати обробку даних, користуючись прикладами, навединими у даній лабораторній
3. Завантажити відповідний *. ipynb файл у classroom

Лабораторна робота № 2

Тема. Кластеризація

Мета. Ознайомитися з методами кластеризації у Sklearn

Теоретичні відомості

Кластеризація - це розбиття множини об'єктів на підмножини (кластери) за заданим критерієм. Кожен кластер включає максимально схожі між собою об'єкти.

Загальноприйнятої класифікації методів немає, але є кілька підходів.

1. **Імовірнісний підхід.** В рамках нього передбачається, що кожен із об'єктів відноситься до одного з класів.

EM-алгоритм застосовується для знаходження оцінок максимальної правдоподібності параметрів імовірнісних моделей (якщо є залежність від прихованих змінних).

K-середніх – алгоритм мінімізує сумарне квадратичне відхилення точок кластерів від їхніх центрів.

Алгоритми сімейства FOREL засновані на ідеї об'єднання об'єктів в один кластер в областях їхньої максимальної концентрації.

2. **Підходи з урахуванням систем штучного інтелекту**. Велика умовна група методів відрізняється від методичної точки зору.

Метод нечіткої кластеризації С-середніх передбачає розбиття наявної множини елементів на кілька нечітких множин. Метод є удосконаленим варіантом К-середніх.

Нейронна мережа Кохонена - клас нейронних мереж із шаром Кохонена, що складається з лінійних формальних нейронів.

Генетичний алгоритм - алгоритм пошуку, який застосовується для вирішення задач оптимізації та моделювання за допомогою випадкового підбору, варіації та комбінування шуканих параметрів. Використовуються механізми, аналогічні до природного відбору в природі.

4. **Ієрархічний підхід**. Передбачає наявність вкладених груп кластерів різного порядку. Виділяються агломеративні та дивізіонні (об'єднувальні та розділяючі) алгоритми. Залежно кількості ознак можуть виділятися політетичні (використовують при порівнянні кількох ознак одночасно) і монотетичні (використовують при застосуванні однієї ознаки) методи класифікації.

Як описати кластеризацію формально?

У кластеризації мають справу з безліччю об'єктів (X) та безліччю номерів кластерів (Y). Задано функцію відстані між об'єктами (p). Потрібно розбити навчальну вибірку на кластери, щоб кожен кластер складався з об'єктів, близьких за метрикою p, а об'єкти різних кластерів істотно відрізнялися. У цьому кожному об'єкту приписується номер кластера y(i). **Алгоритм кластеризації** - це функція, яка будь-якому об'єкту X ставить у відповідність номер кластера Y.

Хід роботи

Завантаження даних

1.Завантажити датасет за посиланням:
https://archive.ics.uci.edu/ml/datasets/iris
Дані представлені як data файла. Дані являють собою інформацію про три класи квітів

2. Створити Python скрипт. Завантажити дані в датафрейм

```
import pandas as pd
import numpy as np
data = pd.read_csv('iris.data',header=None
```

K-середніх

1. Проведемо кластеризацію методів k-середніх

```
from sklearn.cluster import KMeans
```

```
k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
k_means.fit(no_labeled_data)
```

2. Отримаємо центри кластерів і визначимо які спостереження до якого кластеру потрапили

```
from sklearn.metrics.pairwise import pairwise_distances_argmin
k_means_cluster_centers = k_means.cluster_centers_
k_means_labels = pairwise_distances_argmin(no_labeled_data,
k_means_cluster_centers)
```

3.Построим результаты классификации для признаков попарно (1 и 2, 2 и 3, 3 и 4)

```
import matplotlib.pyplot as plt
f, ax = plt.subplots(1, 3)
colors = ['#4EACC5', '#FF9C34', '#4E9A06']
print(ax)
for i in range(3):
    my_members = k_means_labels == i
    cluster_center = k_means_cluster_centers[i]
    for j in range(3):
        ax[j].plot(no_labeled_data[my_members, j],
no_labeled_data[my_members, j+1], 'w',
markerfacecolor=colors[i], marker='o', markersize=4)
         ax[j].plot(cluster_center[j],
cluster_center[j+1], 'o',
markerfacecolor=colors[i],
         markeredgecolor='k', markersize=8)
plt.show()
```

Опишіть отримані результати. За якими ознаками став найкращий розподіл. Як впливає значення параметра *n_init.*

4. Зменшіть розмірність даних до 2 використовуючи метод головних компонентів і намалюйте карту для всієї області значень, на якій кожен кластер займає певну область зі своїм кольором (як це робити).

5. Дослідіть роботу алгоритму k-середніх за різних параметрів *init*. Спочатку потрібно виконати кілька разів із параметрів 'random', потім для вручну вибраних точок.

6. Визначте найкращу кількість методом ліктя.

7. Проведіть кластеризацію за допомогою пакетної кластеризації k-середніх. У чому відмінність від звичайного методу k-середніх. Побудуйте діаграму розсіювання, на якій буде виділено точки, які для різних методів потрапили до різних кластерів.

Ієрархічна кластеризація

1.Проведемо ієрархічну кластеризацію на тих самих даних

```
from sklearn.cluster import AgglomerativeClustering
hier = AgglomerativeClustering(n_clusters=3, linkage='average')
hier = hier.fit(no_labeled_data)
hier_labels = hier.labels_
```

2. Відобразимо результати кластеризації

```
f, ax = plt.subplots(1, 3)
colors = ['#4EACC5', '#FF9C34', '#4E9A06']
for i in range(3):
     my_members = hier_labels == i
     for j in range(3):
         ax[j].plot(no_labeled_data[my_members, j],
no_labeled_data[my_members, j+1], 'w',
         markerfacecolor=colors[i], marker='o', markersize=4)
plt.show()
```

У чому відмінність від методу k-середніх

3.Проведіть дослідження для різних розмірів кластерів (від 2 до 5). Наведіть отримані результати

4. Намалюйте дендограму рівня 6

5. Згенеруйте випадкові дані у вигляді двох кілець

```
import random
import math
data1 = np.zeros([250,2])
for i in range(250):
    r = random.uniform(1, 3)
    a = random.uniform(0, 2 * math.pi)
    data1[i,0] = r * math.sin(a)
    data1[i,1] = r * math.cos(a)
data2 = np.zeros([500,2])
for i in range(500):
    r = random.uniform(5, 9)
    a = random.uniform(0, 2 * math.pi)
    data2[i,0] = r * math.sin(a)
    data2[i,1] = r * math.cos(a)
data = np.vstack((data1, data2))
```

6. Проведіть ієрархічну кластеризацію

```
hier = AgglomerativeClustering(n_clusters=2, linkage='ward')
hier = hier.fit(data)
hier_labels = hier.labels_
```

7. Виведіть отримані результати

```
my_members = hier_labels == 0
plt.plot(data[my_members, 0], data[my_members, 1], 'w',
marker='o',
markersize=4,
color='red',linestyle='None')
my_members = hier_labels == 1
```

```
plt.plot(data[my_members, 0], data[my_members, 1], 'w',
marker='o',
markersize=4,
color='blue',linestyle='None')
plt.show()
```

8.Досліджуйте кластеризацію за всіх параметрів *linkage*. Відобразіть та обґрунтуйте отримані результати. Для яких випадків, який тип зв'язку працює найкраще.

ЗАВДАННЯ

1. Вибрати набір даних, користуючись порталом https://www.kaggle.com/
2. Виконати обробку даних, користуючись прикладами, навединими у даній лабораторній
3. Завантажити відповідний *. ipynb файл у classroom

Лабораторна робота №3

Тема. Кластеризація (DBSCAN, OPTICS)

Мета. Ознайомлення з методами кластеризації *Sklearn*

Теоретичні відомості

Кластеризація DBSCAN

Алгоритм кластеризації DBSCAN у машинному навчанні з використанням Python DBSCAN означає просторову кластеризацію на основі щільності для шумових програм. Це – алгоритм кластеризації без учителя, який використовується для пошуку базових вибірок із високою щільністю для розширення кластерів. У цій статті я познайомлю вас із кластеризацією DBSCAN у машинному навчанні з використанням Python. Алгоритм кластеризації DBSCAN заснований на концепції зразків ядра, неосновних зразків і викидів:

Зразки ядра: зразки, представлені в області з високою щільністю, мають мінімальну кількість точок вибору з радіусом eps.
Зразки не ядра: зразки близькі до зразків ядра, але не є зразками ядра, а дуже близькі до зразків ядра. Зразки не ядра лежать у радіусі eps від зразків ядра, але в них немає мінімальних точок відбору проб.
Викиди: зразки, які не є частиною зразків ядра, і зразки не ядра, і знаходяться далеко від усіх зразків.
Алгоритм кластеризації DBSCAN працює добре, якщо всі кластери досить щільні та добре представлені з низькою щільністю.

Упорядкування точок виявлення кластерної структури (англ. Ordering points to identify the clustering structure, OPTICS) — це алгоритм знаходження кластерів у просторових даних з урахуванням щільності. Основна ідея алгоритму схожа на DBSCAN, але алгоритм призначений для позбавлення однієї з головних слабкостей алгоритму DBSCAN —

проблеми виявлення змістовних кластерів у даних, що мають різні щільності. Щоб це зробити, точки бази даних (лінійно) упорядковуються так, що просторово близькі точки стають сусідніми в упорядкуванні. Крім того, для кожної точки запам'ятовується спеціальна відстань, що представляє густину, яку слід прийняти для кластера, щоб точки належали одному кластеру. Це можна представити у вигляді дендрограми.

Хід роботи

Завантаження даних

1. Завантажити набір даних за посиланням: https://www.kaggle.com/arjunbhasin2013/ccdata.

Дані представлені у вигляді файлу csv. Набір даних містить пропущені значення.

2. Створити скрипт Python. Завантажити дані в датафрейм, видаливши стовбці з мітками і відкинувши дані з пропущеними значеннями.

```
import pandas as pd
import numpy as np
data = pd.read_csv('CC GENERAL.csv').iloc[:,1:].dropna()
```

DBSCAN

1. Проведемо кластеризацію методів к-средних

```
from sklearn.cluster import KMeans
k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
k_means.fit(no_labeled_data)
```

2. Оскільки різні ознаки лежать в різних шкалах, то стандартизуємо дані

```
from sklearn import preprocessing
data = np.array(data, dtype='float')
min_max_scaler = preprocessing.StandardScaler()
scaled_data = min_max_scaler.fit_transform(data)
```

3.Проведемо кластеризацію методів DBSCAN за параметрами за замовчуванням. Виведемо мітки кластерів, кількість кластерів, а також відсоток спостережень, які кластеризувати не вдалося

```
clustering = DBSCAN().fit(scaled_data)
print(set(clustering.labels_))
print(len(set(clustering.labels_)) - 1)
print(list(clustering.labels_).count(-1) /
len(list(clustering.labels_)))
```

Опишіть усі параметри, які приймає DBSCAN.

4. Побудуйте графік кількості кластерів та відсотка не кластеризованих спостережень залежно від максимальної розглянутої дистанції між спостереженнями. Мінімальне значення кількості точок, що утворюють, кластер залишити за замовчуванням.

5. Побудуйте графік кількості кластерів та відсотка не кластеризованих спостережень залежно від мінімального значення кількості точок, що утворюють кластер. Максимальну дистанцію, що розглядається між спостереженнями, залиште за замовчуванням.

6. Визначте значення параметрів, за яких кількість кластерів виходить від 5 до 7, і відсоток не кластеризованих спостережень не перевищує 12%.

7. Зменште розмірність даних до 2 за допомогою методу основних компонент. Візуалізуйте результати кластеризації, отримані в пункті 6 (мітки повинні бути отримані на даних до зменшення розмірності). гайд із візуалізації: https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py

OPTICS

1.Опишіть параметри методу OPTICS:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html#sklearn.cluster.OPTICS , , а також якими атрибутами він володіє

2. Знайдіть такі параметри методу OPTICS (*max_eps *і min_samples) при яких, щоб отримати результати, близькі до результатів DBSCAN з точки 6

В чому відмінність від методу OPTICS від методу DBSCAN

3. Візуалізуйте отриманий результат, а також побудуйте графік досягнень (досяжна ділянка) гайд: https://scikit-

learn.org/stable/auto_examples/cluster/plot_optics.html#sphx-glr-auto-examples-cluster-plot-optics-py

4. Досліджуйте роботу методу OPTICS з використанням різних метрик (виберіть не менше 5 метрик)

ЗАВДАННЯ

1. Вибрати набір даних, користуючись порталом

https://www.kaggle.com/

2. Виконати обробку даних, користуючись прикладами, навединими у даній лабораторній

3. Завантажити відповідний *. ipynb файл у classroom

Лабораторна робота №4

Тема. Класифікація (Байєсовські методи, дерева)

Мета. Ознайомлення з методами класифікації *Sklearn*

Теоретичні відомості

Бейсовський підхід - це найбільш академічний погляд на машинне навчання, який має переваги та недоліки

Перевага методу: дає чіткий математичний опис навчання та чисельні оцінки достовірності гіпотез. Тоді як звичайні підходи найчастіше видають лише одну гіпотезу, як достовірну. Наприклад, в результаті навчання виходить цілком певна нейронна мережа (набір ваг). І у нас немає надійної оцінки того, на скільки ця мережа краща за якусь іншу. Ми навіть не можемо сказати, чи справді вона найкраща.

Недолік методу у тому, що в реальному житті не завжди можна реалізувати бездоганну математичну модель. Але за допомогою математичної моделі все одно можна отримати певну оцінку якості навчання.

Насправді, будь-які підходи до машинного навчання подібні. У нас завжди є дві основні речі:

По-перше, це набір гіпотез. Це може бути набір функцій, або набір ваг для нейронів нейронної мережі, або набір всіляких вирішальних дерев…

По-друге, у нас є набір навчальних даних.

Наше завдання завжди зводиться до одного: визначити, яка з наших моделей найбільш адекватна нашим навчальним даним.

Байєсовський підхід дає точний чисельний критерій вирішення цього завдання.

З класичною ймовірністю ми часто маємо справу у фізичних та інженерних завданнях. Тут все просто, треба поставити багато експериментів і поділити кількість успіхів на кількість експериментів.

Це частотне розуміння ймовірності.

Скажімо, якщо ви кинули монетку 100 разів і 49 разів випав "орел", то можна говорити, що ймовірність випадання "орла" близька до 49% (що більше експериментів, то точніше ми оцінимо ймовірність).

Тобто тут у нас є модель і ми хочемо підрахувати ймовірність будь-якого результату. Це пряме завдання.

Модель "монетка" дуже проста: у неї дві сторони і вона не може стати на ребро.

Але в комп'ютерному навчанні найчастіше зустрічаються зовсім інші завдання: коли ви не знаєте модель і за відомою поведінкою треба побудувати модель. Точніше, строго кажучи, визначити, яка з моделей найбільш ймовірна, тому що точно відповісти ви зазвичай не можете.

Теорема Байєса дуже проста. Вона є очевидною тотожністю, проте це не применшує її важливість.

Нагадаю позначення.

• $p(h)$ — ймовірність того, що відбувається подія h (h, у даному випадку — не число, а позначення певної абстракції, яка, втім, може бути і числом; ми позначатимемо букою h конкретну гіпотезу)

$p(a \wedge b)$ — ймовірність того, що сталося і подія a, і подія b

$p(a \mid b)$ — ймовірність того, що сталося a за умови, що відбувалося b (тобто, якщо a — володіння червоною машиною, а b — володіння машиною, то $p(a \mid b)$ — ймовірність володіння червоною машиною, розрахована тільки для автовласників)

Теорема Байєса є наслідком очевидного твердження

$$p(a \wedge b) = p(a \mid b)\, p(b) = p(b \mid a)\, p(a)$$

тобто

$$p(a \mid b) = \frac{p(b \mid a)\, p(a)}{p(b)}$$

Отже $p(b)$ має бути більше 0, якщо b - це неймовірна подія то і $p(a \mid b)$ також неймовірне.

Застосування теореми Байєса до машинного навчання

Давайте позначимо наші дані – D, а наші гіпотези – h. Тоді нам треба знайти ймовірність гіпотези для наших даних $p(h \mid D)$, яка за теоремою Байєса дорівнює:

$$p(h \mid D) = \frac{p(D \mid h)\, p(h)}{p(D)}$$

Нас цікавить тільки співвідношення ймовірностей, тому ми можемо викинути з цього виразу $p(D)$ (D не залежить від h) і $p(h)$ (вважатимемо, що всі гіпотези рівноймовірні; строго кажучи, це не завжди так, ми ще повернемося до це питання, але у багатьох випадках це справедливо).

Виходить, нам треба знайти таку гіпотезу h, на яку максимально $p(D \mid h)$.

Говорячи математичною мовою, ми повинні кожної гіпотези обчислити її апостеріорну ймовірність і вибрати гіпотезу, на яку ця ймовірність

максимальна. Цей підхід називається *maximum a posteriori probability* (MAP). Якщо суворо дотримуватися його, то ви отримаєте найкращу гіпотезу. І це математично підтверджено. На жаль, повне проходження MAP можливе далеко не завжди. Наприклад, даних дуже багато. Або рішення треба дати швидко. Тому в реальному житті використовуються різні модифікації MAP, але ми не вдаватимемося в це.

Хід роботи

Завантаження даних

Завантажити датасет за посиланням:

https://archive.ics.uci.edu/ml/datasets/iris .

Дані представлені як data файла. Дані являють собою інформацію про трьох класах квітів

2. Створити Python скрипт. Завантажити дані в датафрейм

```
import pandas as pd
import numpy as np
data = pd.read_csv('iris.data',header=None)
```

3. Виділимо дані та їх мітки

```
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()
```

4. Перетворюємо тексти міток до чисел

```
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)
```

5. Розбиваємо вибірку на навчальну та тестову

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.5)
```

Байєсівські методи

1. Проведемо класифікацію спостережень наївським байєсівським методом:

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

```
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum()) #кількість спостережень, які були неправильно визначені
```

Опишіть атрибути цього класифікатора

2. Використовуючи функцію score(), виведіть точність класифікації

3. Побудуйте графік залежності неправильно класифікованих спостережень та точності класифікації від розміру тестової вибірки. Розмір тестової вибірки змінюйте від 0,05 до 0,95 з кроком 0,05. Параметр random_state зробіть рівним номеру залікової книжки. Обґрунтуйте отримані результати

4. Проведіть класифікацію за допомогою посилань:

MultinomialNB: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

ComplementNB: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB

BernoulliNB: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB

Опишіть особливості методів

Класифікуючі дерева

1.Класифікація за допомогою дерев на тих самих даних

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
```

2. Використовуючи функцію `score()`, виведіть точність класифікації

3. Виведіть характеристики дерева, кількість листя та глибину, використовуючи функції `get_n_leaves` та `get_depth`

4. Виведіть зображення отриманого дерева

```
import matplotlib.pyplot as plt
plt.subplots(1,1,figsize = (10,10))
tree.plot_tree(clf, filled = True)
plt.show()
```

Опишіть отриманий рисунок

5. Побудуйте графік залежності неправильно класифікованих спостережень та точності класифікації від розміру тестової вибірки. Розмір тестової вибірки змінюйте від 0,05 до 0,95 з кроком 0,05. Параметр random_state зробіть рівним номеру залікової книжки. Обґрунтуйте отримані результати.
6. Дослідіть роботу класифікуючого дерева за різних параметрів *criterion, splitter, max_depth, min_samples_split, min_samples_leaf*

ЗАВДАННЯ

1. Вибрати набір даних, користуючись порталом https://www.kaggle.com/
2. Виконати обробку даних, користуючись прикладами, навединими у даній лабораторній
3. Завантажити відповідний *. ipynb файл у classroom

Лабораторна робота №5

Тема. Класифікація (лінійний дискримінантний аналіз, метод опорних векторів)

Мета. Ознайомлення з методами класифікації *Sklearn*

Теоретичні відомості

Лінійний дискримінантний аналіз (ЛДА, англ. Linear Discriminant Analysis, LDA), нормальний дискримінантний аналіз (англ. Normal Discriminant Analysis, NDA) або аналіз дискримінантних функцій (англ. Discriminant Function Analysis) є узагальненням лінійного дискримінанта Фішера, методу, що використовується розпізнавання образів та машинному навчанні для пошуку лінійної комбінації ознак, яка описує або поділяє два або більше класів чи подій. Комбінація, що вийшла, може бути використана як лінійний класифікатор, або, більш часто, для зниження розмірності перед класифікацією.

ЛДА тісно пов'язані з дисперсійним аналізом (англ. ANalyse Of Variance=ANOVA) і регресійним аналізом, які намагаються висловити одну залежну змінну як лінійної комбінації інших ознак чи вимірів. Проте дисперсійний аналіз використовує якісні незалежні змінні та безперервну залежну змінну, у той час як дискримінантний аналіз має безперервні незалежні змінні та якісну залежну змінну (тобто мітку класу). Логістична регресія та пробіт-регресія більше схожі на ЛДА, ніж дисперсійний аналіз, оскільки вони так само пояснюють якісну змінну через безперервні незалежні змінні. Ці інші методи кращі в додатках, в яких немає резону припускати, що незалежні змінні нормально розподілені, що є фундаментальним припущенням методу ЛДА.

ЛДА тісно пов'язаний також з методом головних компонентів (МГК, англ. Principal Component Analysis, PCA) і факторним аналізом тим, що вони шукають лінійні комбінації змінних, які краще пояснюють дані. ЛДА явно намагається моделювати різницю між класами даних. МГК, з іншого боку, не бере до уваги будь-яку різницю в класах, а факторний аналіз будує комбінації ознак, спираючись швидше на відмінності, а не на подібність. Дискримінантний аналіз відрізняється також від факторного аналізу тим, що не є незалежною технікою — для його роботи має бути визначена різниця між незалежними змінними та залежними змінними (останні також називаються критеріальними змінними).

ЛДА працює, коли виміри, зроблені на незалежних змінних кожного спостереження, є безперервними величинами. Коли маємо справу з якісними незалежними змінними, еквівалентною технікою є дискримінантний аналіз відповідностей.

Дискримінантний аналіз використовується, коли групи відомі апріорі (на відміну кластерного аналізу). Кожен випадок повинен мати значення в одному або кількох заходах кількісного передбачення та значення групової міри. Висловлюючись простими термінами, аналіз дискримінантних функцій є класифікацією, що розбиває об'єкти на групи, класи чи категорії певного типу.

В машинному навчанні **ме́тод опо́рних векторі́в** — це метод аналізу даних для класифікації та регресійного аналізу за допомогою моделей з керованим навчанням з пов'язаними алгоритмами навчання, які називаються **опо́рно-ве́кторними маши́нами** (**ОВМ**, англ. *support vector machines, SVM*, також **опо́рно-ве́кторними мере́жами**, англ. *support vector networks*[1]). Для заданого набору тренувальних зразків, кожен із яких відмічено як належний до однієї чи іншої з двох категорій, алгоритм тренування ОВМ будує модель, яка відносить нові зразки до однієї чи іншої категорії, роблячи це неймовірнісним бінарним лінійним класифікатором. Модель ОВМ є представленням зразків як точок у просторі, відображених таким чином, що зразки з окремих категорій розділено чистою прогалиною, яка є щонайширшою. Нові зразки тоді відображуються до цього ж простору, й робиться передбачення про їхню належність до категорії на основі того, на який бік прогалини вони потрапляють.

На додачу до виконання лінійної класифікації, ОВМ можуть ефективно виконувати нелінійну класифікацію при застосуванні так званого ядрового трюку, неявно відображуючи свої входи до просторів ознак високої вимірності.

Коли дані не є міченими, кероване навчання є неможливим, і виникає необхідність у спонтанному навчанні, яке намагається знайти природне кластерування даних на групи, а потім відображувати нові дані на ці сформовані групи. Алгоритм кластерування, який забезпечує вдосконалення опорно-векторним машинам, називається **опо́рно-ве́кторним кластерува́нням** (англ. *support vector clustering*),[2] і часто використовується в промислових застосуваннях, коли дані або не є міченими, або коли лише деякі дані є міченими як попередня обробка перед проходом класифікації.

Хід роботи

Завантаження даних

1.Завантажити датасет за посиланням:
https://archive.ics.uci.edu/ml/datasets/iris .

Дані представлені як data файла. Дані являють собою інформацію про три класи кольорів

2. Створити Python скрипт. Завантажити дані в датафрейм

```
import pandas as pd
import numpy as np
data = pd.read_csv('iris.data',header=None)
```

3. Виділимо дані та їх мітки

```
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()
```

4.Перетворюємо тексти міток у числа

```
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)
```

5. Розіб'ємо вибірку на навчальну та тестову

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.5)
```

Лінійний дискримінантний аналіз

1.Проведемо класифікацію спостережень використовуючи LDA:

https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.5,random_state=0)
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum()) #кількість спостережень,
які були неправильно визначені
```

Опишіть атрибути та параметри цього класифікатора

2. Використовуючи функцію score(), виведіть точність класифікації

3. Побудуйте графік залежності неправильно класифікованих спостережень та точності класифікації від розміру тестової вибірки. Розмір тестової вибірки

змінюйте від 0,05 до 0,95 з кроком 0,05. Параметр random_state зробіть рівним номеру залікової книжки. Обґрунтуйте отримані результати.

4. Опишіть навіщо потрібна функція transform? Застосуйте її і візуалізуйте результати.

5. Досліджуйте роботу класифікатора за різних параметрів solver, shrinkage.

6. Задайте апріорну ймовірність класу з номером 1 рівну 0.7, решті класів задайте рівні апріорні ймовірності. Як це далося взнаки на результаті?

Метод опорних векторів

1.Реалізувати класифікацію з використанням SVM на заданому наборі даних

https://scikit-learn.org/stable/modules/svm.html#classification

```
clf = svm.SVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X, Y))
```

2.Визначіть точність класифікації, з використанням функції `score()`

3.Виведіть наступну інформацію

```
print(clf.support_vectors_)
print(clf.support_)
print(clf.n_support_)
```

Поясніть, що відображають ці параметри, від чого залежать.

4. Побудуйте графік залежності неправильно класифікованих спостережень та точності класифікації від розміру тестової вибірки Розмір тестової вибірки змінюйте від 0,05 до 0,95 з кроком 0,05. Параметр *random_state* зробіть рівним номеру своєї залікової книжки. Обґрунтуйте отримані результати.

5. Дослідіть роботу методу опорних векторів за різних значень *kernel, degree, max_iter*

6. Проведіть дослідження для методів

NuSV: https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC

та

LinearSVC: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

Сформулюйте відмінності від SVC.

ЗАВДАННЯ

1. Вибрати набір даних, користуючись порталом

https://www.kaggle.com/

2. Виконати обробку даних, користуючись прикладами, навединими у даній лабораторній

3. Завантажити відповідний *. ipynb файл у classroom

Лабораторна робота № 6

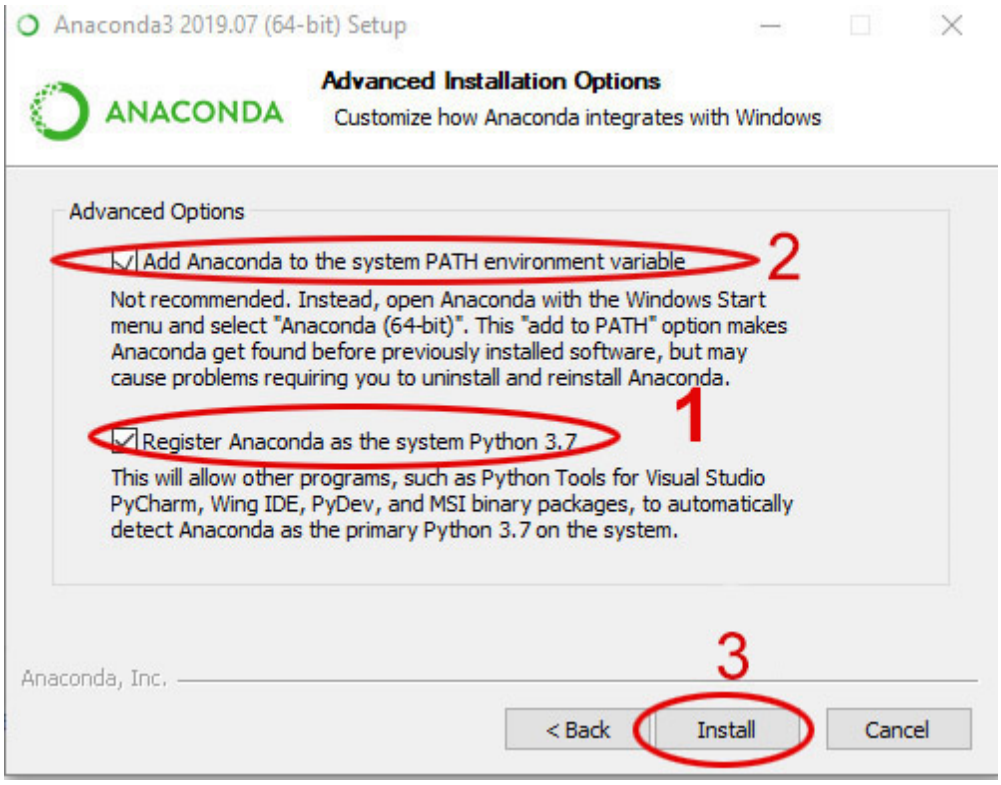
Тема: «Бібліотека TensorFlow для навчання глибоких нейронних мереж»
Мета. У цій роботі необхідно дізнатися, як TensorFlow може бути використаний для побудови нейронних мереж з метою класифікації.

TensorFlow - це бібліотека машинного навчання з відкритим кодом для досліджень та виробництва. TensorFlow пропонує API для початківців та експертів для розробки для ПК, мобільних додатків, веб-сайтів та хмари. Почнемо роботу з TensorFlow покроково.

Установка TensorFlow

1.Завантажити дистрибутив «**Anaconda**» з сайту за посиланням **https://www.anaconda.com/distribution/**

При інсталяції встановити таке налаштування:



2. Відкрити консоль «**Anaconda Prompt**» в

3. Інсталювати TensorFlow

```
conda install tensorflow
```

4 Перейти в папку з jupyter зошитами до цих лабораторних.
5. В рядку зі шляхом до папки набрати «cmd»+Enter
6. В консолі набрати Jupyter notebook + Enter

Початкова підготовка

Keras – це бібліотека високого рівня забезпечує будівельні блоки для створення та навчання моделей глибокого навчання. Її використовують для швидкого створення проектів, для виконання розширених досліджень та створення програмних продуктів з трьома ключовими перевагами:

1.Зручність для користувачів.
Keras має простий, стійкий інтерфейс, оптимізований для випадків загального використання. Він забезпечує чітку реакцію на помилки користувача.

2.Властивість модульності та здатність до композиції
Моделі Keras виготовляються шляхом з'єднання налаштованих будівельних блоків разом з невеликими обмеженнями.

3. Здатність до легкого розширення
Існує можливіть створити спеціальні будівельні блоки, які виражають нові ідеї при дослідженні. Можна додавати нові шари, функції втрат та розробляти сучасні моделі.

Імпортування `tf.keras`
Це API високого рівня призначене для створення та тенування моделей, що включає першокласну підтримку функціоналу, характерного для TensorFlow, такого як «гарячий» запуск, пайплайн tf.data, та оцінювачі. `tf.keras` робить TensorFlow простішим у використанні без шкоди для гнучкості та продуктивності.
Для початку імпортуйте tf.keras як частину налаштування програми TensorFlow:

```
!pip install -q pyyaml  # Required to save models in YAML format
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
from tensorflow.keras import layers

print(tf.version.VERSION)
print(tf.keras.__version__)
```

```
Результат:
1.14.0
2.2.4-tf
```

tf.keras може запускати будь-який сумісний з Керас код, але пам’ятайте, що версія tf.keras в останньому випуску TensorFlow може не збігатися з останньою версією keras з PyPI. Перевіряйте tf.keras .__ version__.

Ключові посилання:
tf.keras
Keras API specification.

Побудова найпростішої моделі

Послідовна модель.

У Keras необхідно об’єднувати шари нейронної мережі для побудови моделі. Модель - це, як правило, граф шарів. Найпоширеніший тип моделі - це стек шарів: модель tf.keras.Sequences.

Виконуючи таке об’єднання будуємо просту повнозв’язну мережу (тобто багатошаровий перцептрон):

```
model = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model:
model.add(layers.Dense(64, activation='relu'))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add a softmax layer with 10 output units:
model.add(layers.Dense(10, activation='softmax'))
```

```
Результат:

WARNING: Logging before flag parsing goes to stderr.
W0903 18:52:34.494595  4124 deprecation.py:506] From
C:\Users\MICHAEL\Anaconda3\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling
VarianceScaling.__init__ (from
tensorflow.python.ops.init_ops) with dtype is deprecated
and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead
of passing it to the constructor
```

Ключові посилання:
tf.keras.Sequential

Налаштування шарів

Доступно багато tf.keras.layers з деякими загальними параметрами конструктора:
1.*активація*: встановлення функції активації шару. Цей параметр задається назвою вбудованої функції або як об'єкт, що викликається. За замовчуванням активація не застосовується.
2.kernel_initializer та bias_initializer (*ініціалізація ядра та зсуву*): схеми ініціалізації, які створюють ваги шару (ядро та зміщення). Цей параметр - це ім'я або об'єкт, який можна викликати. Це за замовчуванням маємо ініціалізатор "Glorot uniform".
3.kernel_regularizer та bias_regularizer: схеми регуляризації, для ваг шару (ядро та зміщення), такі як L1 або L2 регуляризація. За замовчуванням не застосовується регуляризація.

Наступний код створює об’єкт tf.keras.layers.Dense за допомогою аргументів конструктора:

```
# Create a sigmoid layer:
layers.Dense(64, activation='sigmoid')
# Or:
layers.Dense(64, activation=tf.sigmoid)

# A linear layer with L1 regularization of factor 0.01
applied to the kernel matrix:
layers.Dense(64,
kernel_regularizer=tf.keras.regularizers.l1(0.01))

# A linear layer with L2 regularization of factor 0.01
applied to the bias vector:
layers.Dense(64,
bias_regularizer=tf.keras.regularizers.l2(0.01))

# A linear layer with a kernel initialized to a random
orthogonal matrix:
layers.Dense(64, kernel_initializer='orthogonal')

# A linear layer with a bias vector initialized to 2.0s:
layers.Dense(64,
bias_initializer=tf.keras.initializers.constant(2.0))
```

Ключові посилання:
tf.keras.layers
tf.keras.layers.Dense

Тренування та оцінка

Налаштування тренування

Після побудови моделі налаштовуємо її процес навчання, викликаючи метод компіляції:

```
model = tf.keras.Sequential([
# Adds a densely-connected layer with 64 units to the
model:
layers.Dense(64, activation='relu', input_shape=(32,)),
# Add another:
layers.Dense(64, activation='relu'),
# Add a softmax layer with 10 output units:
layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.train.AdamOptimizer(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

`tf.keras.Model.compile` має три важливі аргументи:

1.*оптимізатор*: Цей об'єкт визначає процедуру навчання. Передаємо екземпляри оптимізатора з модуля `tf.train`, наприклад такі: `tf.train.AdamOptimizer`, `tf.train.RMSPropOptimizer` або `tf.train.GradientDescentOptimizer`.
2.*функція втрат*: функція мінімізації під час оптимізації. Поширені варіанти включають середню квадратичну `помилку (mse)`, `categorical_crossentropy` і `binary_crossentropy`. Функції втрати задаються назвою або передачею об'єкта, що викликається, з модуля `tf.keras.losses`.
3. *метрики*: використовується для моніторингу тренування. Це імена рядків або виклики з модуля tf.keras.metrics.

Далі показано кілька прикладів налаштування моделі для тренування:

```
# Configure a model for mean-squared error regression.
model.compile(optimizer=tf.train.AdamOptimizer(0.01),
              loss='mse',       # mean squared error
              metrics=['mae'])  # mean absolute error

# Configure a model for categorical classification.
model.compile(optimizer=tf.train.RMSPropOptimizer(0.01),
              loss=tf.keras.losses.categorical_crossentro
py,
```

```
              metrics=[tf.keras.metrics.categorical_accur
acy])
```

Ключові посилання:
tf.keras.Model.compile
tf.train
tf.train.AdamOptimizer
tf.train.RMSPropOptimizer
tf.train.GradientDescentOptimizer
tf.keras.losses
tf.keras.metrics

Ввід даних NumPy

Для невеликих наборів даних використаємо масиви NumPy в пам'яті для підготовки та оцінки моделі. Модель налаштовуємо до даних тренувань за допомогою методу fit:

```
import numpy as np

def random_one_hot_labels(shape):
  n, n_class = shape
  classes = np.random.randint(0, n_class, n)
  labels = np.zeros((n, n_class))
  labels[np.arange(n), classes] = 1
  return labels

data = np.random.random((1000, 32))
labels = random_one_hot_labels((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32)
```

tf.keras.Model.fit має три важливі аргументи:

1.*Епохи*: Тренування структурується в епохи. Епоха - це одна ітерація над усіма вхідними даними (вони формують менші партії обробки).
2. batch_size (розмір партії): Коли передаються дані NumPy, модель розбиває дані на менші партії та ітерує ці партії під час тренування. Ціле число визначає розмір кожної партії. Майте на увазі, що остання партія може бути меншою, якщо загальна кількість даних не ділиться на розмір партії.
3.validation_data (Дані оцінки): Під час тренування моделі можливо легко контролювати її ефективність на деяких даних перевірки. Передача цього аргументу - кортеж входів та міток - дозволяє моделі відображати втрати та метрики в режимі виводу для переданих даних у кінці кожної епохи.

Ось приклад використання validation_data:

```
import numpy as np

data = np.random.random((1000, 32))
labels = random_one_hot_labels((1000, 10))

val_data = np.random.random((100, 32))
val_labels = random_one_hot_labels((100, 10))

model.fit(data, labels, epochs=10, batch_size=32,
          validation_data=(val_data, val_labels))
```

Ключові посилання:
[tf.keras.Model.fit]

Ввід наборів `tf.data`

Використаєм API наборів даних для масштабування великих наборів даних або тренувань на декількох пристроях. Передаємо екземпляр tf.data.Dataset методу fit:

```
# Instantiates a toy dataset instance:
dataset = tf.data.Dataset.from_tensor_slices((data,
labels))
dataset = dataset.batch(32)
dataset = dataset.repeat()

# Don't forget to specify `steps_per_epoch` when calling
`fit` on a dataset.
model.fit(dataset, epochs=10, steps_per_epoch=30)
```

Тут метод `fit` використовує аргумент `steps_per_epoch` - це кількість навчальних кроків, які модель виконує перед тим, як перейти до наступної епохи. Оскільки набір даних складається партій даних, для даного фрагмента не потрібно batch_size.

Набори даних також можуть бути використані для перевірки:

```
dataset = tf.data.Dataset.from_tensor_slices((data,
labels))
dataset = dataset.batch(32).repeat()

val_dataset =
```

```
tf.data.Dataset.from_tensor_slices((val_data,
val_labels))
val_dataset = val_dataset.batch(32).repeat()

model.fit(dataset, epochs=10, steps_per_epoch=30,
          validation_data=val_dataset,
          validation_steps=3)
```

Ключові посилання:
tf.data.Dataset

Оцінка і прогноз

Методи `tf.keras.Model.evaluate` і `tf.keras.Model.predict` можуть використовувати дані `NumPy` та `tf.data.Dataset`.

Для оцінки втрат і метрик застосовуємо код:

```
data = np.random.random((1000, 32))
labels = random_one_hot_labels((1000, 10))

model.evaluate(data, labels, batch_size=32)

model.evaluate(dataset, steps=30)
```

Використовуєм вихід останнього шару для висновку по наданих даних, як масив NumPy:

```
result = model.predict(data, batch_size=32)
print(result.shape)
```

Ключові посилання:
tf.keras.Model.evaluate
tf.keras.Model.predict

Тренування нейронної мережі: основна класифікація

Початкові завантаження

Цей матеріал містить інструкції для навчання моделі нейронної мережі для класифікації зображень одягу.

Наведемо огляд повної програми TensorFlow з деталями, які пояснимо в процесі розгляду.

Використаємо tf.keras, API високого рівня для створення та навчання моделей у TensorFlow.

```
from __future__ import absolute_import, division,
print_function, unicode_literals
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)
```

Завантаження Fashion MNIST

Будемо використовувати набір даних Fashion MNIST, який містить 70 000 зображень у відтінках сірого в 10 категоріях. На зображеннях представлені окремі предмети одягу з низькою роздільною здатністю (28 на 28 пікселів), як видно з рисунку:

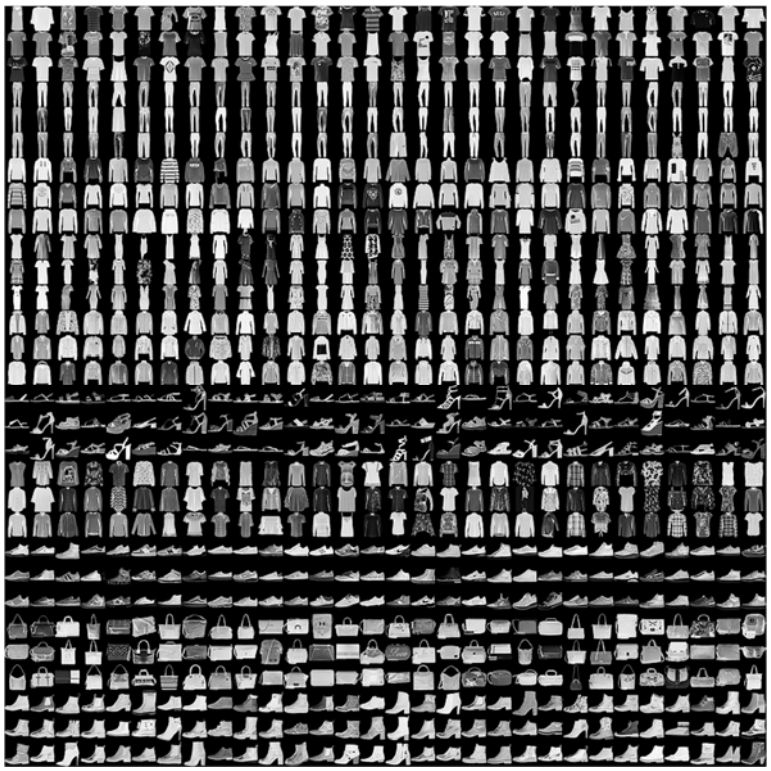


Рис. 1 Приклад зображень з файлу Fashion MNIST

Fashion MNIST є заміною для класичного набору даних MNIST. Він часто використовується як "Hello, World"у програмах машинного навчання для комп'ютерного зору. Набір даних MNIST містить зображення рукописних цифр (0, 1, 2 тощо) в тому ж форматі, що і предмети одягу, які ми тут використовуватимемо.

Застосування Fashion MNIST є дещо складніша завданням, ніж звичайний MNIST. Обидва набори даних порівняно невеликі і використовуються для перевірки того, чи працює алгоритм так, як очікувалося. Вони хороші вихідні точки для тестування та налагодження коду.

Ми будемо використовувати 60 000 зображень для тренування мережі та 10 000 зображень, щоб оцінити, наскільки точно мережа навчилася класифікувати зображення. Ми можемо отримати доступ до Fashion MNIST безпосередньо з TensorFlow, просто імпортуючи та завантажуючи дані:

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels)
= fashion_mnist.load_data()
```

Дане завантаження набору даних повертає чотири масиви NumPy:
1.Масиви train_images та train_labels утворюють навчальний набір. Це дані, які модель використовує для вивчення.

2.Модель тестується на тестовому наборі, що складається з масивів test_images та test_labels.

Зображення представляють собою масиви NumPy розмірністю 28x28 пікселів з діапазоном зміни значення пікселя від 0 до 255. Мітки - це масив цілих чисел, які можуть приймати значення в діапазоні 0 до 9. Вони відповідають класу одягу, який показано на кожному з зображень. У таблиці показано відповідність між номером мітки

Мітка	Клас одягу
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Зберігаємо класи міток

Кожному зображенню відповідає одна мітка. Оскільки назви класів не включені до набору даних, збережемо їх, щоб використовувати при побудові зображень:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover',
'Dress', 'Coat','Sandal', 'Shirt', 'Sneaker', 'Bag',
'Ankle boot']
```

Дослідження даних

Розглянемо формат набору даних перед навчанням моделі. Наведені нижче дані показують, що у навчальному наборі є 60 000 зображень, а кожне зображення має розмір 28 x 28 пікселів:

```
train_images.shape
```

Відповідно у навчальному наборі є 60 000 міток:

```
len(train_labels)
```

Кожна мітка - це ціле число від 0 до 9:

```
train_labels
```

У тестовому наборі є 10 000 зображень. Знову ж таки, кожне зображення представлене у вигляді 28 x 28 пікселів:

```
test_images.shape
```

Тестовий набір міток містить відповідно 10 000 міток зображень:

```
len(test_labels)
```

Попередня обробка даних

Попередньо обробимо дані перед навчанням мережі. Якщо оглянути довільне зображення у навчальному наборі, то можна побачити, що значення пікселів знаходяться в діапазоні від 0 до 255:

```
plt.figure()
plt.imshow(train_images[5])
plt.colorbar()
plt.grid(False)
plt.show()
```

Ми нормалізуємо ці значення до діапазону від 0 до 1 перед завантаженням в модель нейронної мережі. Для цього ми ділимо значення на 255. Важливо, щоб навчальний набір і тестовий набір були попередньо оброблені однаково:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Відобразимо перші 25 зображень із навчального набору з відображенням назви класу під кожним зображенням. Це дасть нам можливість переконатися, що дані у правильному форматі, і ми готові створити та навчити мережу.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Побудова моделі

Для того, щоб побудувати модель необхідно задати шари, налаштувати їх та виконати компіляцію моделі.

Задавання шарів

Шар є основним будівельним блоком нейронної мережі. Шари виконують добування корисної інформації з даних, що надходять у них.
Здебільшого глибоке навчання складається зі з'єднання простих шарів. Більшість шарів, таких як `tf.keras.layers.Dense`, мають параметри, які змінюються під час тренувань.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Перший шар у цій мережі, tf.keras.layers.Flatten, перетворює формат зображень з 2d-масиву (розміром 28 на 28 пікселів) у 1d-масив розміром 28 * 28 = 784 пікселів. Цей шар можемо розглядати як послідовну конкатенацію рядків пікселів на зображенні. Цей шар не має параметрів для вивчення; він лише переформатує дані.

Після вирівнювання пікселів мережа складається з послідовності двох шарів tf.keras.layers.Dense. Це повністю зв'язані нейронні шари. Перший шар має 128 вузлів (або нейронів). Другий (і останній) шар - це 10-вузловий softmax-шар. Останній повертає масив з 10 балів ймовірності, який дорівнює 1. Кожен вузол містить бал, який вказує на ймовірність того, що поточне зображення належить одному з 10 класів.

Компіляція моделі

Перш ніж модель будемо вважати готовою до навчання, необхідно виконати ще кілька налаштувань. Вони додаються під час кроку компіляції моделі:

1.*Функція втрат*. Ця функція дає можливість виміряти наскільки точною є модель на етапі тренувань. Ми хочемо мінімізувати цю функцію, щоб «спрямовувати» модель в потрібному напрямку.

2. *Оптимізатор* - це те, як модель оновлюється на основі даних, які вона бачить, і її функція втрат.

3.*Метрики* - використовується для контролю кроків навчання та тестування. У наступному прикладі використовується точність, фракція зображень, які правильно класифіковані.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Тренування моделі
Тренування моделі нейронної мережі вимагає наступних кроків:

1.Завантаження тренувальних даних в модель. У нашому прикладі дані представлені масивами `train_images` та `train_labels`.
2. Модель навчається співставляти зображення та мітки.
3.Ми ставимо завдання для моделі зробити передбачення щодо тестового набору. У цьому прикладі це масив test_images. Ми перевіряємо наскільки прогнози моделі відповідають міткам з масиву test_labels.

Щоб розпочати навчання необхідно викликати модель model.fit . Модель буде налаштовуватися на тренувальний набір даних :

```
model.fit(train_images, train_labels, epochs=5)
```

Під час тренування моделі відображаються показники втрат та точності. Ця модель досягає точності приблизно 0,89 (або 89%) за даними тренувань.

Оцінка точності

Далі порівняємо, як працює модель на тестовому наборі даних:

```
test_loss, test_acc = model.evaluate(test_images,
test_labels)
print('Test accuracy:', test_acc)
```

Тепер можемо порівняти, як працює модель на тестовому наборі даних і навчальному наборі даних.
Виявляється, точність на тестовому наборі даних трохи менша (87%), ніж точність на навчальному наборі даних (89%). Цей розрив між точністю тренувань та точністю тесту є прикладом перенавчання. Перенавчання - це коли модель працює на нових даних гірше, ніж на тренувальних даних.

Прогнозування за допомогою моделі

Якщо модель натренована, то тоді ми можемо використовувати її для прогнозування щодо деяких зображень.

```
predictions = model.predict(test_images)
```

Попередній код задає для моделі інструкцію передбачення мітки для кожного зображення в тестовому наборі. Давайте подивимось на прогнози:

```
predictions[0]

predictions[100]

predictions[9000]
```

Прогноз представлений масивом з 10 чисел. Вони описують «впевненість» моделі, що образ відповідає кожному з 10 різних предметів одягу. Ми можемо побачити, яка мітка має найвище значення довіри:

```
np.argmax(predictions[0])
```

Тож модель найбільш впевнена, що це зображення - це ankle boot, або клас 9. І ми можемо перевірити тестову мітку, щоб переконатися, що це правильно:

```
test_labels[0]
```

Ми можемо представити графічно прогноз на один з 10 класів для кожного зображення

```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img =
predictions_array[i], true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}%
({})".format(class_names[predicted_label],
                                100*np.max(predictions_ar
ray),

                                class_names[true_label]),
                                color=color)
```

```
def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i],
true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array,
color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Давайте розглянемо 0-е зображення, прогнози та масив передбачень.

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions,  test_labels)
plt.show()
```

12-е зображення

```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions,  test_labels)
plt.show()
```

Давайте побудуємо кілька зображень із їх прогнозами. Правильні мітки прогнозування - сині, а неправильні мітки прогнозування - червоні. Цифра дає відсоток (із 100) для передбачуваної мітки. Зауважте, що це може бути помилково, навіть коли дуже впевнено.

```
# Plot the first X test images, their predicted label,
and the true label
# Color correct predictions in blue, incorrect
predictions in red
num_rows = 5
num_cols = 3
```

```
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)
plt.show()
```

Нарешті, використовуємо навчену модель, щоб зробити передбачення щодо одного зображення

```
# Grab an image from the test dataset
img = test_images[0]

print(img.shape)
```

Моделі `tf.keras` оптимізовані для того, щоб робити прогнози на партію чи на набір прикладів одразу. Отже, хоча ми використовуємо одне зображення, нам потрібно додати його до списку:

```
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))
print(img.shape)
```

Тепер передбачаємо зображення:

```
predictions_single = model.predict(img)
print(predictions_single)

plot_value_array(0, predictions_single, test_labels)
plt.xticks(range(10), class_names, rotation=45)
plt.show()
```

model.predict повертає список списків, по одному для кожного зображення в пакеті даних. Візьміть прогнози для нашого (одиничного) зображення в партії:

```
prediction_result = np.argmax(predictions_single[0])
print(prediction_result)
```

Код, що використовується для початкової підготовки наведено у файлі Jupyter-Notebook «Note2. ipynb»

Завдання до лабораторної роботи №

1.Налаштувати роботу, запустивши код, що використовується для початкової підготовки, який наведено у файлі Jupyter-Notebook «Note1. ipynb». На даному прикладі вивчити основні принципи роботи TensorFlow.

2.На прикладі Fashion MNIST налаштувати навчання моделі класифікації

3.Використовуючи формат `Fashion MNIST`, сформувати набір даних для моделі класифікації. Використати підходящий набір з сайту

https://archive.ics.uci.edu/ml/datasets.php

4.Створити модель для розпізнавання латинських символів та виконати дослідження даної моделі.

5.У звіті представити jupter notebook моделей з п.1-3.

Лабораторна робота № 7

Тема: «Основи застосування повнозв'язних мереж для задач регресії

Мета. У цій роботі необхідно дізнатися, як TensorFlow може бути використаний для побудови нейронних мереж з метою реалізації регресійних задач.

У регресійній задачі ми передбачаємо вихід неперервної величини, як ціна чи ймовірність. Порівнюймо цю задачу з задачею класифікації, у якій ми вибираємо клас із списку класів (наприклад, задача визначення по малюнку яблука чи апельсина).

Цей матеріал використовує класичний набір даних Auto MPG і створює модель для прогнозування ефективності використання палива автомобілів кінця 1970-х та початку 1980-х років. Для цього ми задаєм в моделі опис великої кількості автомобілів того часу. Цей опис включає такі ознаки, як: циліндри, переміщення, кінські сили та вага.

У цьому прикладі використовується API tf.keras.

```
# Use seaborn for pairplot
!pip install -q seaborn
from __future__ import absolute_import, division,
print_function, unicode_literals

import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

Набір даних Auto MPG

Цей набір даних доступний на UCI Machine Learning Repository.

Отримання даних

Завантажуємо набір даних.

```
dataset_path = keras.utils.get_file("auto-mpg.data",
"http://archive.ics.uci.edu/ml/machine-learning-
databases/auto-mpg/auto-mpg.data")
dataset_path
```

Імпортуємо набір даних, використовуючи pandas

```
column_names =
['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(dataset_path,
names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
```

Очистка даних

Набір даних містить певну кількість невідомих даних

```
dataset.isna().sum()
```

Стовпчик "Origin" визначає категорію, а не число. Зробимо перетвореннння:

```
origin = dataset.pop('Origin')

dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0
dataset.tail()
```

Розбиття набору даних на тестовий і тренувальний

Розділимо набір даних на навчальний набір і тестовий набір.
Ми будемо використовувати тестовий набір у підсумковій оцінці нашої моделі.

```
train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

Перевірка даних

Швидко ознайомимось із спільним розподілом кількох пар стовпців із навчального набору.

```
sns.pairplot(train_dataset[["MPG", "Cylinders",
"Displacement", "Weight"]], diag_kind="kde")
plt.show()
```

Також подивимось на загальну статистику:

```
train_stats = train_dataset.describe()
train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats
```

Розділимо ознаки за мітками

Відокремте цільове значення або "мітку" від озенак. Ця мітка - це значення, яке ми маємо навчити модель передбачати.

```
train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')
```

Нормалізація даних

Подивимось ще раз на train_stats block та зазначимо, наскільки різні діапазони кожної ознаки.

Доброю практикою є нормалізація функцій, які використовують різні масштаби та діапазони. Хоча модель може збігатися без нормалізації ознак, це робить навчання складнішим, і це робить отриману модель залежною від вибору одиниць, що використовуються на вході.

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

Нормовані дані ми будемо використовувати для тренування моделі.

Побудова моделі

Давайте побудуємо нашу модель. Тут ми будемо використовувати послідовну модель з двома щільно з'єднаними прихованими шарами та вихідним шаром, який повертає єдине безперервне значення. Етапи побудови моделі загорнуті у функцію build_model.

```
def build_model():
  model = keras.Sequential([
    layers.Dense(64, activation=tf.nn.relu,
```

```
input_shape=[len(train_dataset.keys())]),
    layers.Dense(64, activation=tf.nn.relu),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mean_squared_error',
                optimizer=optimizer,
                metrics=['mean_absolute_error',
'mean_squared_error'])
  return model
model = build_model()
```

Перевірка моделі

Використовуватимемо метод `.summary`, щоб надрукувати простий опис моделі:

```
model.summary()
```

Тепер перевіримо модель. Для цього використаємо зріз з 10 прикладів з навчальних даних та викличемо на модель.

```
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result
```

Якщо ваша модель працює і дає результат очікуваної форми та типу, то перейдемо до тренувань

Тренування моделі

Будемо навчати модель з використанням 1000 епох і запишемо точність навчання та перевірки достовірності в об’єкт історії.

```
# Display training progress by printing a single dot for
each completed epoch
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 1000
```

```
history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])
```

Візуалізуємо хід навчання моделі, використовуючи статистику, що зберігається в об'єкті історії.

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

```
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [MPG]')
  plt.plot(hist['epoch'], hist['mean_absolute_error'],
           label='Train Error')
  plt.plot(hist['epoch'],
hist['val_mean_absolute_error'],
           label = 'Val Error')
  plt.ylim([0,5])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$MPG^2$]')
  plt.plot(hist['epoch'], hist['mean_squared_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'],
           label = 'Val Error')
  plt.ylim([0,20])
  plt.legend()
  plt.show()


plot_history(history)
```

Отриманий графік має показувати незначне поліпшення або навіть деградацію помилки перевірки приблизно через 100 епох.
Спробуємо оновити виклик `model.fit`, щоб автоматично припинити навчання, коли оцінка перевірки не покращиться. Ми будемо використовувати зворотний

виклик EarlyStopping, який перевіряє умови навчання для кожної епохи. Якщо задана кількість епох проходить, не проявляючи поліпшення, то автоматично будемо припиняти навчання.

```
model = build_model()

# The patience parameter is the amount of epochs to check
for improvement
early_stop =
keras.callbacks.EarlyStopping(monitor='val_loss',
patience=10)

history = model.fit(normed_train_data, train_labels,
epochs=EPOCHS,
                    validation_split = 0.2, verbose=0,
callbacks=[early_stop, PrintDot()])

plot_history(history)
```

На знову отриманому графіку видно, що наборі перевірки середня помилка зазвичай становить приблизно +/- 2 MPG.
Розглянемо питання наскільки добре модель узагальнюється за допомогою тестового набору, який ми не використовували при навчанні моделі. Це говорить, наскільки добре ми можемо очікувати, що модель спрогнозує, коли ми будемо використовувати її в реальному світі.

```
loss, mae, mse = model.evaluate(normed_test_data,
test_labels, verbose=0)

print("Testing set Mean Abs Error: {:5.2f}
MPG".format(mae))
```

Виконання прогнозів

Зробимо прогноз значення MPG, використовуючи дані в тестовому наборі:

```
test_predictions =
model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
```

```
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```

Якщо, модель налаштована правильно, то вона маэ добре прогнозувати. На завершення розглянемо розподіл помилок.

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
_ = plt.ylabel("Count")
```

Звичайно, при такій кількості даних ми не можемо очікувати гаусівського розподілу. Але певна тенденція повинна проглядатися

Завдання до лабораторної роботи №7

1.Налаштувати роботу, запустивши код, що використовується для початкової підготовки, який наведено у файлі Jupyter-Notebook «Note1. ipynb». На даному прикладі вивчити основні принципи роботи TensorFlow.

2.На прикладі `Auto MPG` налаштувати навчання регресійної моделі.

3.Використовуючи формат `Auto MPG`, сформувати набір даних для регресійної моделі, використовуючи набір з набору

https://archive.ics.uci.edu/ml/datasets.php

4.У звіті представити jupter notebook моделей з п.1-3.