Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №1

з дисципліни «Системи штучного інтелекту»

на тему

«Моделі машинного навчання»

Виконали:
студентки групи ІС-12
Павлова Софія
Гоголь Софія

Викладач:
Коломоєць С. О.

Київ – 2023

1. Постановка задачі

Мета: ознайомитись з принципами функціонування, створення, навчання та використання моделей машинного навчання.

Завдання:

Для обраної задачі класифікації (або регресії) на основі типового датасету створити модель машинного навчання, навчити її на датасеті, перевірити результат на тестовій вибірці.

22	Wine Quality(Регресія)	Одношаровий персептрон

2. Виконання

2.1. Датасет та задача

Відповідно по варіанту, ми використали датасет **Wine Quality**.

Опис датасету: https://www.tensorflow.org/datasets/catalog/wine_quality?hl=en.

Сайт, з якого було завантажено датасет: https://archive.ics.uci.edu/dataset/186/wine+quality.

У датасет включені два набори даних, пов'язані із зразками червоного та білого вина vinho verde, з півночі Португалії.

Вхідні дані включають об'єктивні тести (наприклад, значення pH), а результати засновані на органолептичних даних (медіана принаймні 3 оцінок, зроблених експертами з винної справи). Кожен експерт оцінив якість вина від 0 (дуже погано) до 10 (відмінно).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	fixed acid	volatile ac	citric acid	residual si	chlorides	free sulfu	total sulfu	density	pH	sulphates	alcohol	quality	
2	7	0.27	0.36	20.Лип	0.045	45	170	1.001	3	0.45	08.Сер	6	
3	06.Бер	0.3	0.34	01.Чер	0.049	14	132	0.994	03.Бер	0.49	09.Тра	6	
4	08.Січ	0.28	0.4	06.Вер	0.05	30	97	0.9951	Бер.26	0.44	10.Січ	6	
5	07.Лют	0.23	0.32	08.Тра	0.058	47	186	0.9956	Бер.19	0.4	09.Вер	6	
6	07.Лют	0.23	0.32	08.Тра	0.058	47	186	0.9956	Бер.19	0.4	09.Вер	6	
7	08.Січ	0.28	0.4	06.Вер	0.05	30	97	0.9951	Бер.26	0.44	10.Січ	6	
8	06.Лют	0.32	0.16	7	0.045	30	136	0.9949	Бер.18	0.47	09.Чер	6	
9	7	0.27	0.36	20.Лип	0.045	45	170	1.001	3	0.45	08.Сер	6	
10	06.Бер	0.3	0.34	01.Чер	0.049	14	132	0.994	03.Бер	0.49	09.Тра	6	
11	08.Січ	0.22	0.43	01.Тра	0.044	28	129	0.9938	Бер.22	0.45	11	6	
12	08.Січ	0.27	0.41	Січ.45	0.033	11	63	0.9908	Лют.99	0.56	12	5	
13	08.Чер	0.23	0.4	04.Лют	0.035	17	109	0.9947	Бер.14	0.53	09.Лип	5	
14	07.Вер	0.18	0.37	01.Лют	0.04	16	75	0.992	Бер.18	0.63	10.Сер	5	
15	06.Чер	0.16	0.4	01.Тра	0.044	48	143	0.9912	Бер.54	0.52	12.Кві	7	

Рисунок 1 – Таблиці вхідних даних

Задача регресії полягає в **прогнозуванні числового значення змінної на основі вхідних даних**.

Метриками точності створеної моделі для вирішення задачі регрійного аналізу є **середньоквадратична помилка** (Mean Squared Error, MSE), **середньоабсолютна помилка** (Mean Absolute Error, MAE) та **коефіцієнт детермінації** (R-squared, R^2).

Так, як вхідні дані у нас обмежені, а їх об’єм невеликий, дану задачу регресії можна оцінити як просту і зробити припущення, що для заданого датасету найліпше підійде різновид лінійної моделі.

2.2. Завантаження датасету та первинна обробка даних

Спершу спробуємо виконати задачу регресії на основі **моделі опорного вектора**.

З огляду на те, що наш датасет має багато ознак, що подаються на вхід нашій моделі машинного навчання і нам доведеться багато працювати окремо зі стовпцями, було прийнято рішення використати структуру даних **DataFrame**.

Так, як набір даних включає 2 файли, напишемо програму таким чином, щоб при запуску, користувач міг обирати, з якого файлу він хоче зчитати дані.

Лістинг коду:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from seaborn import pairplot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
# Ігнорування warnings
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

files = ['winequality-white.csv', 'winequality-red.csv']

# Вибір режиму зчитування даних
print('Оберіть джерело вхідних даних та подальші дії:')
for i in range(len(files)):
    print(i + 1, '-', files[i])
data_mode = int(input('mode:'))

# Якщо джерело даних існує
if data_mode in range(1, len(files) + 1):
    # Завантаження датасету
    file = files[data_mode - 1]
    data = pd.read_csv(file, sep=';')

    # Вигляд датасету
    print('-----------------------------')
    print(file, ':')
    print('-----------------------------')
    print(data.head(5), '\n')
```

Результат:

```
Оберіть джерело вхідних даних та подальші дії:
1 - winequality-white.csv
2 - winequality-red.csv
mode:1
----------------------------
winequality-white.csv :
----------------------------
   fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
0            7.0              0.27         0.36  ...       0.45      8.8        6
1            6.3              0.30         0.34  ...       0.49      9.5        6
2            8.1              0.28         0.40  ...       0.44     10.1        6
3            7.2              0.23         0.32  ...       0.40      9.9        6
4            7.2              0.23         0.32  ...       0.40      9.9        6

[5 rows x 12 columns]
```

Рисунок 2 – Вигляд DataFrame, з яким будемо працювати

Розділимо датасет на тренувальні, тесту вальні та валідаційні дані у співвідношенні (75:15:10). Нормалізуємо значення.

Лістинг коду:

```
# Розділ датасету на ознаки - результати тестів вина (X) та цільову змінну - якість вина
(y)
X = data.drop('quality', axis=1)
y = data['quality']

# Розділ даних на тренувальний (75%), тестовий (15%) і валідаційний (10%) набори
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.25,
random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.4,
random_state=42)

# Перевірка співвідношення 75% : 15% : 10%
print('Ознак тренувального датасету:', X_train.shape)
print('Ознак тестового датасету:', X_test.shape)
print('Ознак валідаційного датасету:', X_val.shape, '\n')

# Нормалізація ознак
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

Результат:

```
Ознак тренувального датасету: (3673, 11)
Ознак тестового датасету: (735, 11)
Ознак валідаційного датасету: (490, 11)
```

Рисунок 3 – Обсяги дата сетів, з якими будемо працювати

Виведемо графіки розсіювання ознак датасету за допомогою бібліотеки **seaborn**.

Лістинг коду:

```
# Побудова графіків pairplot
pairplot(data[data.columns], diag_kind='kde')
plt.show()
```

Результат:

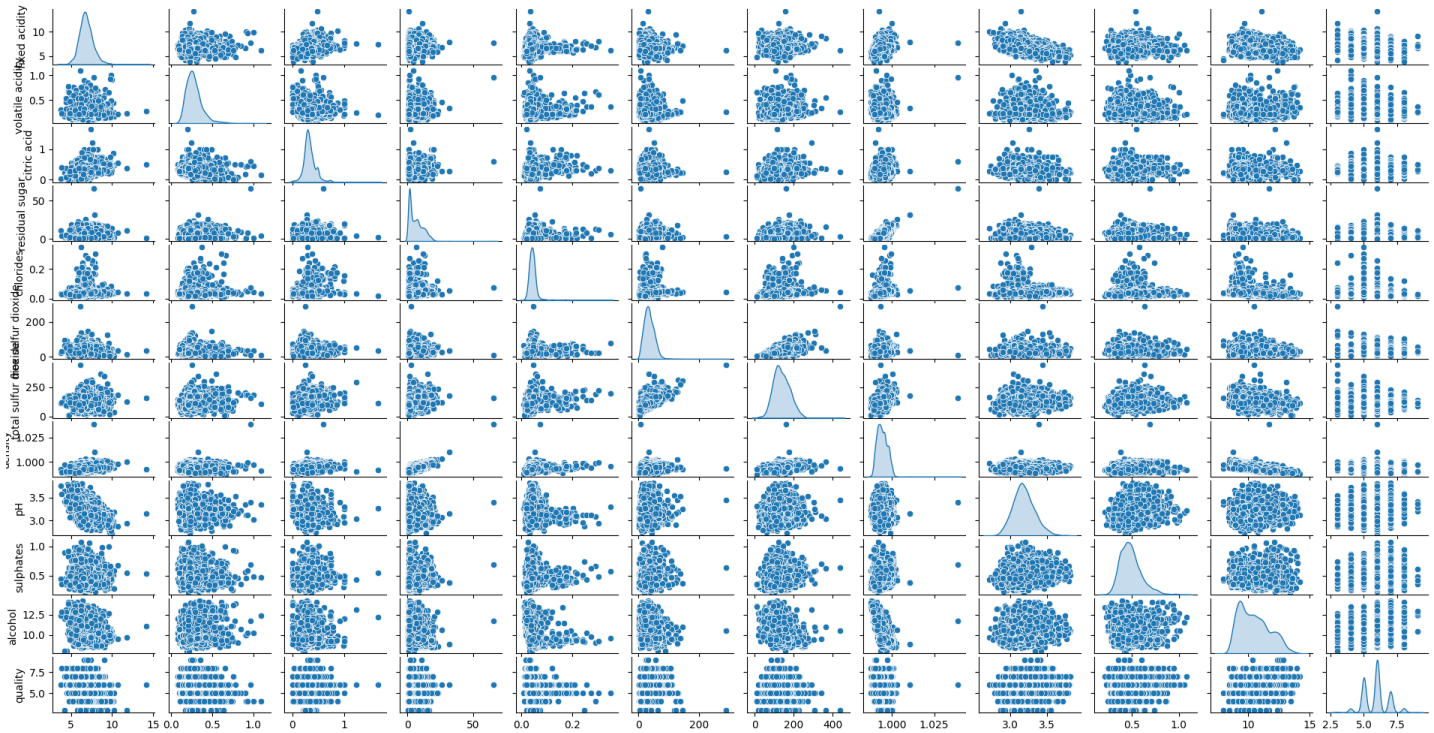

Рисунок 4 – Графіки розсіювання ознак

З графіків бачимо, що дані переважно **не мають лінійної залежності**, що ускладнює процедуру підбору підходящих для обраного датасету моделей.

За допомогою вбудованих функцій бібліотек виведемо опис набору даних.

Лістинг коду:

```
# Опис датасету
print('-----------------------------')
print('Опис:')
print('-----------------------------')
print(data.describe().transpose(), '\n')
```

Результат:

```
----------------------------
Опис:
----------------------------
                      count        mean  ...       75%        max
fixed acidity        4898.0    6.854788  ...    7.3000   14.20000
volatile acidity     4898.0    0.278241  ...    0.3200    1.10000
citric acid          4898.0    0.334192  ...    0.3900    1.66000
residual sugar       4898.0    6.391415  ...    9.9000   65.80000
chlorides            4898.0    0.045772  ...    0.0500    0.34600
free sulfur dioxide  4898.0   35.308085  ...   46.0000  289.00000
total sulfur dioxide 4898.0  138.360657  ...  167.0000  440.00000
density              4898.0    0.994027  ...    0.9961    1.03898
pH                   4898.0    3.188267  ...    3.2800    3.82000
sulphates            4898.0    0.489847  ...    0.5500    1.08000
alcohol              4898.0   10.514267  ...   11.4000   14.20000
quality              4898.0    5.877909  ...    6.0000    9.00000

[12 rows x 8 columns]
```

Рисунок 5 – Опис датасету

2.3. Створення та навчання моделі опорних векторів (SVM)

Використаємо відповідний регресійний SVM-алгоритм.

Лістинг коду:

```
# Створення і навчання регресійної моделі SVM
svm_regressor = SVR(kernel='rbf')  # тип ядра
svm_regressor.fit(X_train, y_train)
```

Для даного алгоритму використовуються наступні типи ядер: Лінійне ядро (Linear Kernel), Поліноміальне ядро (Polynomial Kernel), Радіальне базисне функційне ядро (Radial Basis Function Kernel або RBF Kernel), Сигмоїдне ядро (Sigmoid Kernel) та інші.

Ми обрали тип ядра «**rbf**», оскільки він найкраще себе проявив на нашому датасеті.

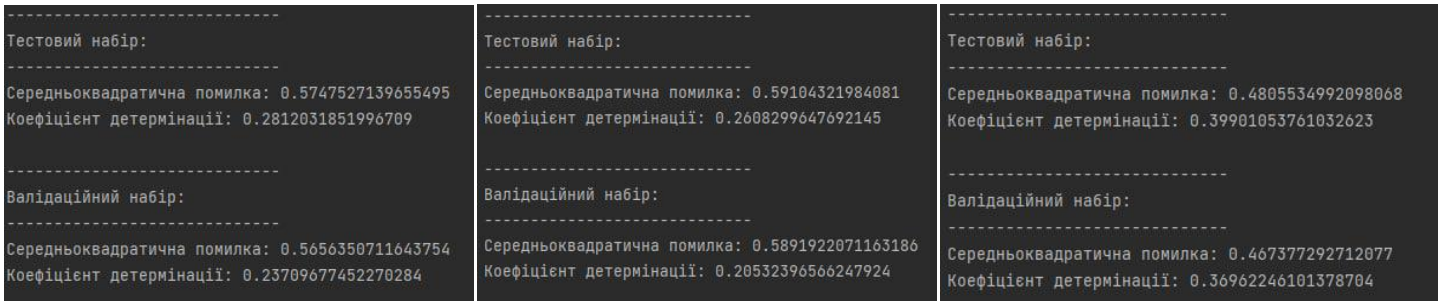

Рисунок 6 – Використання ядер «linear», «poly», «rbf»

2.4. Оцінка точності моделі опорних векторів (SVM)

Оцінимо модель відповідно до висунутих раніше метрик точності на тесту вальному та валідаційному датасетах.

Лістинг коду:

```
# Оцінка на тестовому наборі
y_pred_test = svm_regressor.predict(X_test)
mae_test = np.mean(np.abs(y_test - y_pred_test))
mse_test = mean_squared_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)
print('-----------------------------')
print('Тестовий набір:')
print('-----------------------------')
print('Абсолютна помилка:', mae_test)
print('Середньоквадратична помилка:', mse_test)
print('Коефіцієнт детермінації:', r2_test, '\n')

# Оцінка на валідаційному наборі
y_pred_val = svm_regressor.predict(X_val)
mae_val = np.mean(np.abs(y_val - y_pred_val))
mse_val = mean_squared_error(y_val, y_pred_val)
r2_val = r2_score(y_val, y_pred_val)
print('-----------------------------')
print('Валідаційний набір:')
print('-----------------------------')
print('Абсолютна помилка:', mae_val)
print('Середньоквадратична помилка:', mse_val)
print('Коефіцієнт детермінації:', r2_val)
```

Результат:

```
----------------------------
Тестовий набір:
----------------------------
Абсолютна помилка: 0.5179864115710882
Середньоквадратична помилка: 0.4805534992098068
Коефіцієнт детермінації: 0.39901053761032623

----------------------------
Валідаційний набір:
----------------------------
Абсолютна помилка: 0.5166636336581314
Середньоквадратична помилка: 0.467377292712077
Коефіцієнт детермінації: 0.36962246101378704
```

Рисунок 7 – Точність моделі опорних векторів

2.5. Нейромережа прямого поширення

Спробуємо створити найпростішу нейромережу прямого поширення за допомогою бібліотеки **TensorFlow** та фреймворку **Keras**.

Нормалізуємо ознаки, побудуємо модель та навчимо її на 50 епохах. Оцінимо її точність за допомогою середньо абсолютної помилки. Виведемо графік функції втрат.

Лістинг коду:

```
# Функція побудови моделі linear
def build_and_compile_model_linear(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(units=1)
    ])
    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.1))
    return model

# Функція історії абсолютних помилок
def learning_hist(model, epoch):
    history = model.fit(
        X_train,
        y_train,
        validation_data=(X_test, y_test),
        verbose=0,
        epochs=epoch
    )
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch
    print(hist.tail(), '\n')
    return history
```

```python
# Функція оцінки моделі на валідаційних даних
def val_loss(model):
    val_loss = model.evaluate(X_val, y_val)
    print('Валідаційна абсолютна помилка:', val_loss, '\n')
    return val_loss

# Графік функції втрат
def plot_loss(history, title):
    plt.plot(history.history['loss'], label='Тренувальна')
    plt.plot(history.history['val_loss'], label='Тестова')
    plt.ylim([0, 5])
    plt.xlabel('Епохи')
    plt.ylabel('Помилка оцінки якості вина')
    plt.legend()
    plt.grid(True)
    plt.title(title)

# Нормалізація ознак
    normalizer = tf.keras.layers.Normalization(axis=-1)
    normalizer.adapt(np.array(X_train))

# ----------------------Лінійна регресійна модель з декількома ознаками-----------------

    # Лінійна модель
    linear_model = build_and_compile_model_linear(normalizer)
    # Опис моделі
    linear_model.summary()
    print('')
    # Навчання моделі
    history = learning_hist(linear_model, 50)
    # Функція втрат
    linear_loss = val_loss(linear_model)
    plot_loss(history, 'Лінійна модель')
    plt.show()
    # Збереження результатів
    test_results = {}
    test_results['linear_model'] = linear_loss
```

Результат:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizati  (None, 11)                23
 on)

 dense (Dense)               (None, 1)                 12

=================================================================
Total params: 35 (144.00 Byte)
Trainable params: 12 (48.00 Byte)
Non-trainable params: 23 (96.00 Byte)
_________________________________________________________________
```

```
        loss  val_loss  epoch
45  0.627261  0.700600     45
46  0.627439  0.597356     46
47  0.634032  0.589074     47
48  0.621432  0.619213     48
49  0.642214  0.600150     49

16/16 [==============================] - 0s 2ms/step - loss: 0.5954
Валідаційна абсолютна помилка: 0.5953558683395386
```

Рисунок 8 – Параметри моделі нейромережі прямого поширення та її точність

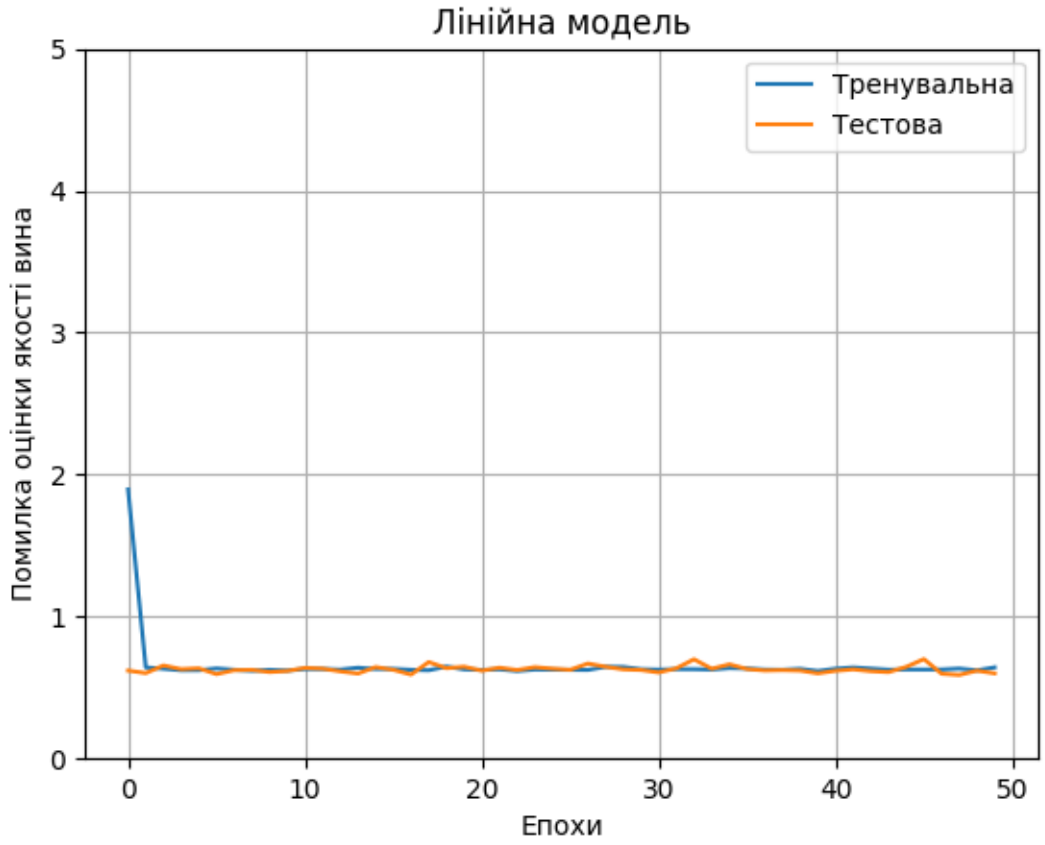

Рисунок 9 – Функція втрат нейромережі прямого поширення

З графіку функції втрат видно, що моделі важко справлятися з обраним датасетом. **Валідаційне значення абсолютної похибки** становить **0.5953558683395386** для **нейромережі прямого поширення**.

2.6. Глибока нейромережа (DNN)

Складнішою альтернативою для розв’язання цієї задачі є створення глибокої нейромережі. Оскільки глибока нейромережа має приховані **Dense** шари й більше параметрів, вона може впоратись із задачею краще.

Аналогічно до нейромережі прямого поширення, створимо, навчимо та протестуємо глибоку нейромережу.

Лістинг коду:

```
# Функція побудови моделі DNN
def build_and_compile_model(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])
    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.001))
    return model

# ----------------------Глибока нейромережа (DNN) з декількома ознаками------------------

    # Глибока нейромережа (DNN)
    dnn_model = build_and_compile_model(normalizer)
    # Опис моделі
    dnn_model.summary()
    print('')
    # Навчання моделі
    history = learning_hist(dnn_model, 100)
    # Функція втрат
    dnn_loss = val_loss(dnn_model)
    plot_loss(history, 'DNN модель')
    plt.show()
    # Збереження результатів
    test_results['dnn_model'] = dnn_loss
    # Порівняння моделей
    print(pd.DataFrame(test_results, index=['Середня абсолютна помилка оцінки якості
вина']).T)
```

Результат:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizati  (None, 11)                23
 on)

 dense_1 (Dense)             (None, 64)                768

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 5016 (19.60 KB)
Trainable params: 4993 (19.50 KB)
Non-trainable params: 23 (96.00 Byte)
_________________________________________________________________

        loss  val_loss  epoch
95  0.423655  0.534222     95
96  0.405837  0.525825     96
97  0.406851  0.544139     97
98  0.408800  0.538518     98
99  0.407325  0.545146     99

16/16 [==============================] - 0s 1ms/step - loss: 0.5550
Валідаційна абсолютна помилка: 0.5550190806388855
```

Рисунок 10 – Параметри моделі глибокої нейромережі та її точність

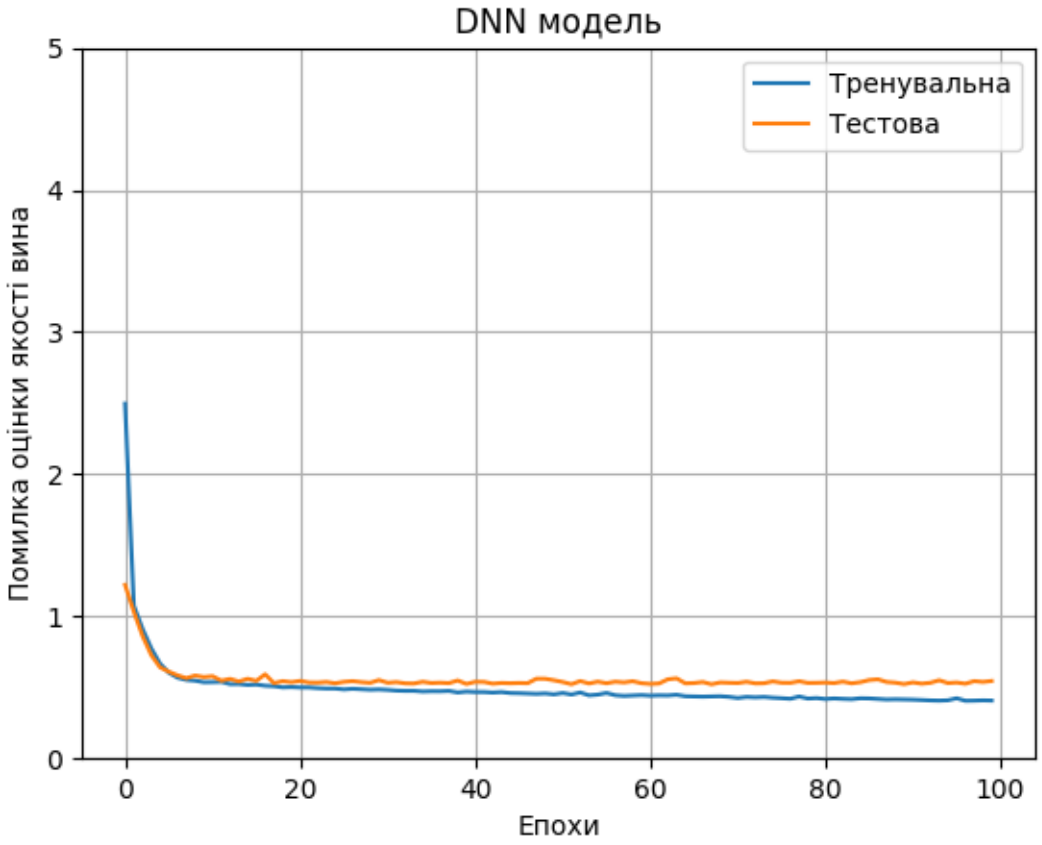

Рисунок 11 – Функція втрат глибокої нейромережі

Бачимо, що дана нейромережа вже краще справляється на обраному датасеті. На графіку спостерігається невелике перенавчання моделі.

Валідаційне значення абсолютної похибки становить **0.5550190806388855** для **глибокої нейромережі**.

2.7. Аналіз отриманих результатів

За отриманими оцінками точності моделей, найкраще впоралась модель опорних векторів. Це можна пояснити особливістю обраного датасету.

Нейромережа простого поширення з лінійною моделлю не підійшла, оскільки ознаки не мали яскраво виражених лінійних залежностей – це добре видно на графіках розподілу.

Натомість моделі з глибокою нейронною мережею можуть бути корисні для складних задач регресії з великою кількістю даних, де необхідно отримати кращу точність завдяки здатності глибокого навчання виявляти складні взаємозв'язки в даних. Наша задача регресії – проста, а взаємозв’язки між даними слабкі, що теж видно з графіку розподілу.

Висновок:

У даній лабораторній роботі ми ознайомилась з принципами функціонування, створення, навчання та використання моделей машинного навчання. Розглянули лінійні та глибокі алгоритми машинного навчання.

Для обраної задачі регресії створили 3 моделі: **опорних векторів, нейромережу прямого поширення** та **глибоку нейромережу**. Навчили їх на тренувальних наборах даних і з’ясували їх точність за допомогою валідаційних датасетів.