

Міністерство освіти і науки України  
Західноукраїнський національний університет

ЗВІТ  
З МОДУЛЬНОЇ РОБОТИ №2  
із дисципліни «Обчислювальний інтелект»  
на тему «Дослідження класифікатора на основі нейронних мереж прямого  
поширення (FeedFoward Neural Networks)»

Виконав:  
Студент: Гадевич Володимир  
Групи: КНм-11

Тернопіль 2024

## Зміст

Вступ.....	3
Хід роботи .....	5
Крок 1 – Читання та нормалізація даних. ....	7
Крок 2 – Створення, навчання та оцінка моделей. ....	9
Крок 3 – Візуалізація результатів. ....	11
Висновок .....	13
Додаток 1. Код програми.....	14

## Вступ

Однією з ключових технологій обчислювального інтелекту є нейронні мережі прямого поширення, або FeedForward Neural Networks. Ці мережі, інспіровані біологічною нейронною мережею людини, дозволяють здійснювати складні обчислення та робити прогнози на основі великої кількості даних. Метою цієї практичної роботи є дослідження класифікатора, побудованого на основі нейронних мереж прямого поширення. Для досягнення цієї мети ми завантажимо базу даних для навчання та тестування класифікатора. Після цього буде проведено дослідження різних параметрів та архітектури нейронних мереж, щоб визначити оптимальну модель для нашої задачі класифікації. Ця робота сприятиме поглибленню розуміння роботи нейронних мереж прямого поширення й дозволить отримати практичний досвід у використанні цих технологій для вирішення реальних завдань класифікації. В результаті, ми отримаємо інструмент, який можна буде застосувати у широкому спектрі областей для забезпечування більш точних та ефективних рішень.

Мета роботи: Дослідження можливостей роботи нейронних мереж прямого поширення для класифікації зображень.

Завдання: Завантажити вхідні дані згідно варіанту, виконати розділення на навчальну та тестову вибірку. Нормалізація даних. Встановлення параметрів навчання нейронних мереж. Дослідження точності навчання із різними архітектурами. Порівняння результатів. Структура нейронних мереж згідно варіанту зображено на рисунку 1.

Вар	Data Set	1-Layer			2-Layers [3,3]			3-Layers [20,7,10]		
		solver	activations	max_iter	solver	activations	max_iter	solver	activations	max_iter
1.	Handwritten Digits	'lbfgs'	'identity'	200	'lbfgs'	('logistic', 'tanh')	200	'lbfgs'	('logistic', 'tanh', 'relu')	200
2.	Cat vs Noncat	'sgd'	'logistic'	300	'sgd'	('logistic', 'relu')	300	'sgd'	('relu', 'logistic', 'relu')	300
3.	Signs	'adam'	'tanh'	100	'adam'	('tanh', 'relu')	100	'adam'	('tanh', 'relu', 'relu')	100
4.	Handwritten Digits	'adam'	'relu'	500	'adam'	('identity', 'relu')	500	'adam'	('identity', 'relu', 'relu')	500
5.	Cat vs Noncat	'lbfgs'	'identity'	200	'lbfgs'	('logistic', 'tanh')	200	'lbfgs'	('logistic', 'tanh', 'relu')	200
6.	Signs	'sgd'	'logistic'	300	'sgd'	('logistic', 'relu')	300	'sgd'	('relu', 'logistic', 'relu')	300
7.	Handwritten Digits	'adam'	'tanh'	100	'adam'	('tanh', 'relu')	100	'adam'	('tanh', 'relu', 'relu')	100

Рисунок 1 - Таблиця із завданнями

Мій варіант відповідає порядковому номеру у списку групи №2.

Відповідно цього варіанту я користуватимусь вхідними даними із файлів «test\_catvnoncat» і «train\_catvnoncat».

Буде розділено на три шари нейронну мережу із максимальною кількістю ітерацій «300» та головним параметром solver буде застосовано «sgd».

## Хід роботи

Перед початком роботи ми повинні встановити набір наступних бібліотек:

1. «h5py»: Ця бібліотека допомагає завантажувати дані з h5 файлів.
2. «numpy»: Використовується для роботи з масивами та математичними операціями.
3. «matplotlib.pyplot»: Вона використовується для створення графіків та візуалізації даних.
4. «sklearn.neural\_network.MLPClassifier»: Ця бібліотека надає класифікатор на основі нейронних мереж.
5. «sklearn.svm.SVC»: Використовується для побудови моделі методом опорних векторів (SVM).
6. «sklearn.model\_selection.train\_test\_split»: Допомагає розділити дані на навчальну та тестову вибірки.
7. «sklearn.metrics.accuracy\_score»: Використовується для обчислення точності моделі.

З цими бібліотеками ми зможемо реалізувати код і провести дослідження нейронних мереж на основі нейронних мереж прямого поширення.

Щоб встановити всі ці бібліотеки одночасно, можна скористатися файлом залежностей requirements.txt, який містить перелік всіх бібліотек та їх версій. Ви можете створити такий файл та додати до нього наступні рядки – рисунок 2.1.

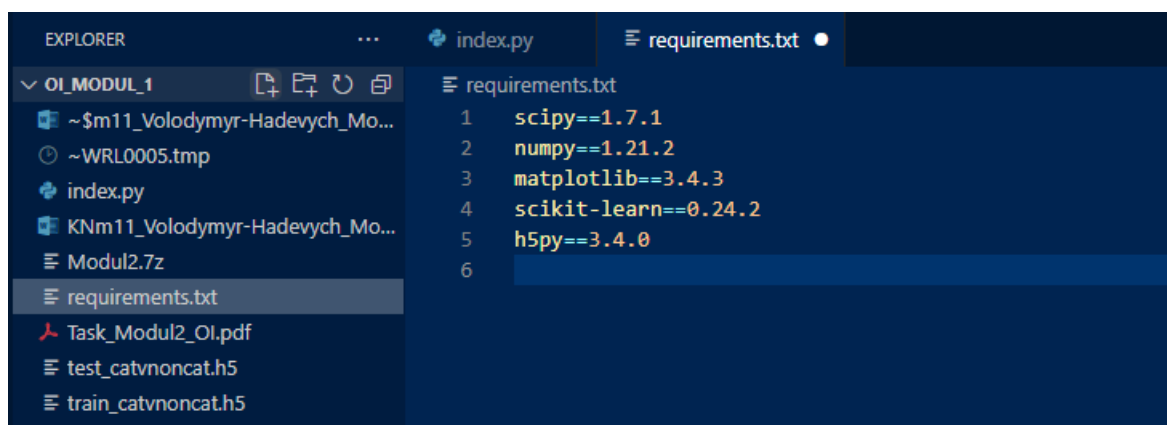


Рисунок 2.1 – Список бібліотек та версій

Після цього можна встановити всі ці бібліотеки одночасно за допомогою команди pip:

```
pip install -r requirements.txt
```

Це автоматично встановить всі зазначені бібліотеки та їх залежності. Такий підхід допомагає уникнути проблем зі сумісністю версій бібліотек та дозволяє легко управляти необхідними залежностями для проекту.

Один із способів перевірки чи встановлено бібліотеки - використати командний рядок або термінал для запуску команди `pip show`, щоб перевірити версії та наявність кожної бібліотеки. Наприклад, для перевірки наявності бібліотеки `scipy` можна ввести команду: `pip show scipy`.

## Крок 1 – Читання та нормалізація даних.

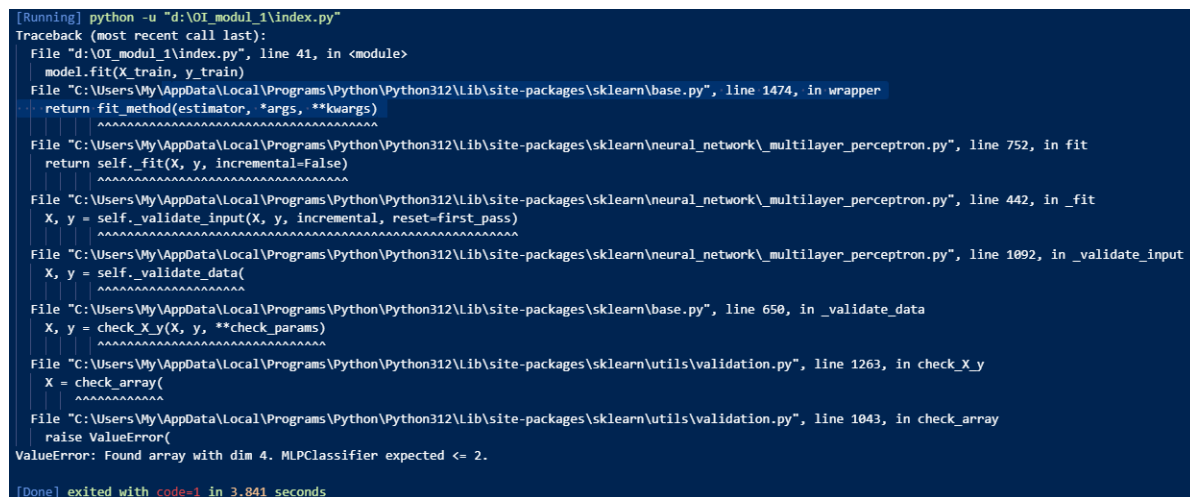
Наступним етапом є зчитування вхідних із файлів із файлами `rain_catvnoncat.h5` - для навчання і `test_catvnoncat.h5` - для тестування. Для цього зазвичай можна було б реалізувати наступний код:

```
# Завантаження даних
with h5py.File('train_catvnoncat.h5', 'r') as train_data:
    X_train = np.array(train_data['train_set_x'])
    y_train = np.array(train_data['train_set_y'])

with h5py.File('test_catvnoncat.h5', 'r') as test_data:
    X_test = np.array(test_data['test_set_x'])
    y_test = np.array(test_data['test_set_y'])

# Нормалізація даних
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Однак при запуску компіляції ми отримуємо складну помилку – рисунок 2.2.



```
[Running] python -u "d:\OI_modul_1\index.py"
Traceback (most recent call last):
  File "d:\OI_modul_1\index.py", line 41, in <module>
    model.fit(X_train, y_train)
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py", line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py", line 752, in fit
    return self._fit(X, y, incremental=False)
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py", line 442, in _fit
    X, y = self._validate_input(X, y, incremental, reset=first_pass)
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py", line 1092, in _validate_input
    X, y = self._validate_data(
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py", line 650, in _validate_data
    X, y = check_X_y(X, y, **check_params)
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py", line 1263, in check_X_y
    X = check_array(
  File "C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py", line 1043, in check_array
    raise ValueError(
ValueError: Found array with dim 4. MLPClassifier expected <= 2.
[Done] exited with code=1 in 3.841 seconds
```

Рисунок 2.2 - Помилки підчас нормалізації даних

Помилка виникає через те, що дані з файлів «`train_catvnoncat.h5`» і «`test_catvnoncat.h5`» мають розмірність 4, а очікується, що вони матимуть розмірність 2. Дані, зазвичай, мають розмірність (кількість прикладів, ширина зображення, висота зображення, канали зображення), але `MLPClassifier` очікує, що вхідні дані матимуть розмірність (кількість прикладів, кількість ознак). Щоб вирішити цю проблему, потрібно змінити форму даних так, щоб вони мали розмірність (кількість прикладів, кількість ознак). Для зображень це означає, що потрібно «розгорнути» їх у одномірний масив.

Наприклад, якщо зображення мають розмірність (кількість прикладів, ширина, висота, канали), то можна використати метод `reshape` для перетворення їх у двовимірний масив, де кожний рядок представлятиме одне зображення. Ось як це можна зробити в вашому коді:

```
# Завантаження даних
with h5py.File('train_catvnoncat.h5', 'r') as train_data:
    X_train = np.array(train_data['train_set_x'])
    y_train = np.array(train_data['train_set_y'])

with h5py.File('test_catvnoncat.h5', 'r') as test_data:
    X_test = np.array(test_data['test_set_x'])
    y_test = np.array(test_data['test_set_y'])

# Перетворення розмірності зображень
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Нормалізація даних
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Це перетворить кожне зображення у масив розміром (кількість ознак). Після цього ваш код повинен працювати коректно.



## Крок 2 – Створення, навчання та оцінка моделей.

На початку нашого коду ми визначаємо моделі для нейронних мереж та метод опорних векторів (SVM). Для цього ми використовуємо класи `MLPClassifier` та `SVC` з бібліотеки `scikit-learn`. Для `MLPClassifier` ми визначаємо різні конфігурації нейронних мереж з різними параметрами, такими як розмірність шарів та активаційні функції. Ось як це виглядає у коді:

```
models_1_layer = [  
    MLPClassifier(hidden_layer_sizes=(1,), solver='sgd', activation='logistic',  
max_iter=300, random_state=42)  
]  
  
models_2_layer = [  
    MLPClassifier(hidden_layer_sizes=(3,), solver='sgd', activation='logistic',  
max_iter=300, random_state=42),  
    MLPClassifier(hidden_layer_sizes=(3,), solver='sgd', activation='relu',  
max_iter=300, random_state=42)  
]  
  
models_3_layer = [  
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd', activation='relu',  
max_iter=300, random_state=42),  
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd',  
activation='logistic', max_iter=300, random_state=42),  
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd', activation='relu',  
max_iter=300, random_state=42)  
]  
  
svm_model = SVC(kernel='linear')
```

Після того, як ми визначили наші моделі, ми навчаємо їх на навчальних даних. Для цього ми використовуємо метод `fit`, який передає навчальні дані `X_train` та відповіді `y_train`. Нижче наведений фрагмент коду, який навчає кожну модель:

```
for model in models_1_layer:  
    model.fit(X_train, y_train)  
  
for model in models_2_layer:  
    model.fit(X_train, y_train)  
  
for model in models_3_layer:  
    model.fit(X_train, y_train)  
  
svm_model.fit(X_train, y_train)
```

Після навчання ми оцінюємо кожну модель на тестових даних, щоб оцінити їх точність передбачень. Ми використовуємо метод `predict`, щоб здійснити передбачення на тестових даних, а потім порівнюємо передбачені значення з фактичними, використовуючи метрику `accuracy_score`. Ось приклад коду для оцінки точності кожної моделі:

```
accuracy = accuracy_score(y_test, model.predict(X_test))
```

Отже, після ініціалізації моделей для нейронних мереж, ми переходимо до кроків створення, навчання та оцінки цих моделей. Спершу, ми створюємо моделі, визначаючи їх архітектуру та параметри, такі як розмірність шарів та активаційні функції. Потім ми навчаємо кожну модель на навчальних даних, використовуючи метод `fit`, щоб модель адаптувалась до вхідних даних та знаходила оптимальні параметри. Після навчання ми оцінюємо кожну модель на тестових даних, порівнюючи їх передбачення з фактичними значеннями та визначаючи точність передбачень за допомогою метрики `accuracy_score`.

Такий підхід до створення, навчання та оцінки моделей надає можливість зробити обґрунтований вибір та отримати найкращі результати в обробці та аналізі даних.

### Крок 3 – Візуалізація результатів.

Завершальним етапом є візуалізація результатів, що дозволяє порівняти ефективність різних моделей та зрозуміти, яка з них найкраще впоралася з поставленою задачею. Це дозволяє визначити оптимальну модель для подальшого використання у реальних задачах. Ми використовуємо бібліотеку Matplotlib для побудови діаграми по результатам коду.

```
plt.bar(labels, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Accuracy of Different Models')
plt.xticks(rotation=45, ha='right')
plt.show()
```

Після виконання цього коду можна побачити стовпчикову діаграму, де кожен стовпчик представляє модель, а його висота відображає точність цієї моделі – рисунок 3.1.

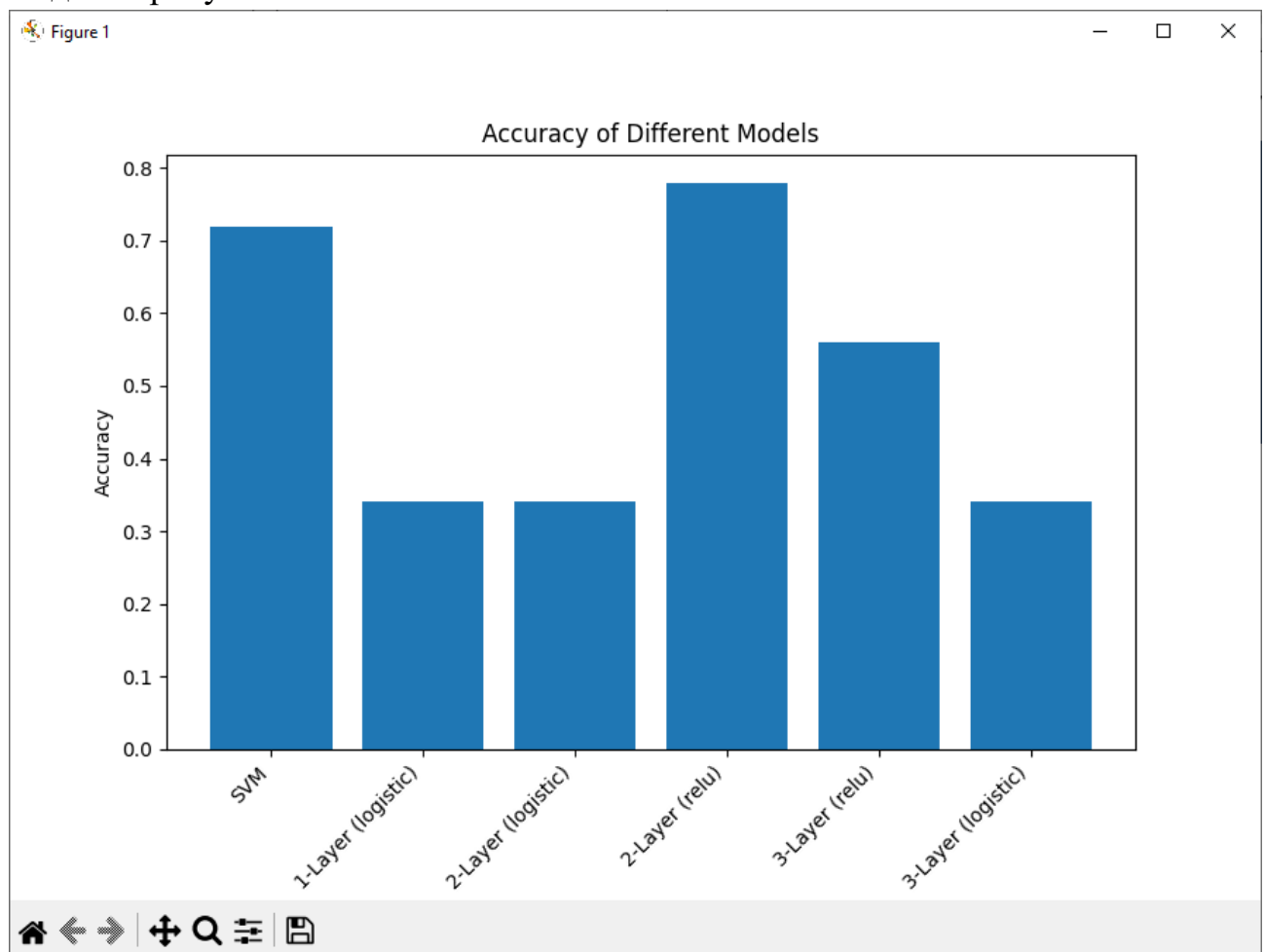


Рисунок 3.1 – Стовпчикова діаграма

На стовпчиковій діаграмі чітко зображено 6 параметрів до порівняння. Проте ми використовували 7 параметрів включаючи SVM.

- 1-Layer має параметр activations 'logistic'
- 2-Layer має параметр activations 'logistic', 'relu'
- 3-Layer має параметр activations 'relu', 'logistic', 'relu'

Для перевірки коректності виконання коду проводимо моніторинг та дублювання результатів кожного шару в термінал – рисунок 3.2.

```
[Running] python -u "d:\OI_modul_1\index.py"
C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
Accuracy for 1-layer model with activation logistic : 0.34
Accuracy for 2-layer model with activation logistic : 0.34
C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
Accuracy for 2-layer model with activation relu : 0.78
C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
Accuracy for 3-layer model with activation relu : 0.56
Accuracy for 3-layer model with activation logistic : 0.34
C:\Users\My\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
Accuracy for 3-layer model with activation relu : 0.56

[Done] exited with code=0 in 409.576 seconds
```

Рисунок 3.2 – Результат коду у терміналі

Як ми пересвідчилися код працює правильно і виводить результат для всіх параметрів 3-Layer, однак оскільки параметр relu дублює свій результат його повторно не включено в діаграму результатів.

## Висновок

В цій роботі ми провели дослідження класифікатора, побудованого на основі нейронних мереж прямого поширення. Було завантажено базу даних для навчання та тестування класифікатора. Проведено дослідження різних параметрів та архітектури нейронних мереж, щоб визначити оптимальну модель для нашої задачі класифікації. Ця робота поглибило розуміння роботи нейронних мереж прямого поширення й дозволило отримати практичний досвід у використанні цих технологій для вирішення реальних завдань класифікації. В результаті, ми отримали робочий інструмент, який можна застосувати у широкому спектрі областей для забезпечування більш точних та ефективних рішень.

## Додаток 1. Код програми

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

with h5py.File('train_catvnoncat.h5', 'r') as train_data:
    X_train = np.array(train_data['train_set_x'])
    y_train = np.array(train_data['train_set_y'])

with h5py.File('test_catvnoncat.h5', 'r') as test_data:
    X_test = np.array(test_data['test_set_x'])
    y_test = np.array(test_data['test_set_y'])

X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

X_train = X_train / 255.0
X_test = X_test / 255.0

models_1_layer = [
    MLPClassifier(hidden_layer_sizes=(1,), solver='sgd', activation='logistic',
max_iter=300, random_state=42)
]

models_2_layer = [
    MLPClassifier(hidden_layer_sizes=(3,), solver='sgd', activation='logistic',
max_iter=300, random_state=42),
    MLPClassifier(hidden_layer_sizes=(3,), solver='sgd', activation='relu',
max_iter=300, random_state=42)
]

models_3_layer = [
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd', activation='relu',
max_iter=300, random_state=42),
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd',
activation='logistic', max_iter=300, random_state=42),
    MLPClassifier(hidden_layer_sizes=(5, 3, 5), solver='sgd', activation='relu',
max_iter=300, random_state=42)
]

for model in models_1_layer:
    model.fit(X_train, y_train)
    accuracy = accuracy_score(y_test, model.predict(X_test))
    print("Accuracy for 1-layer model with activation", model.activation, ":",
accuracy)

for model in models_2_layer:
    model.fit(X_train, y_train)
```

```

    accuracy = accuracy_score(y_test, model.predict(X_test))
    print("Accuracy for 2-layer model with activation", model.activation, ":",
accuracy)

for model in models_3_layer:
    model.fit(X_train, y_train)
    accuracy = accuracy_score(y_test, model.predict(X_test))
    print("Accuracy for 3-layer model with activation", model.activation, ":",
accuracy)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
svm_accuracy = accuracy_score(y_test, svm_model.predict(X_test))

labels = ['SVM', '1-Layer (logistic)', '2-Layer (logistic)', '2-Layer (relu)', '3-
Layer (relu)', '3-Layer (logistic)', '3-Layer (relu)']
accuracies = [svm_accuracy] + [accuracy_score(y_test, model.predict(X_test)) for
model in models_1_layer + models_2_layer + models_3_layer]
plt.bar(labels, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Accuracy of Different Models')
plt.xticks(rotation=45, ha='right')
plt.show()

```