**ПРАКТИЧНА РОБОТА №3**

**Бінарна класифікація**

Мета: Закріпити основні поняття використання моделі бінарної класифікації в машинному навчанні.

Завдання:

1. Ознайомитеся з набором даних, що містить вимірювання, отримані з зображень двох видів турецького рису.
2. Створите бінарний класифікатор для розподілу зерен рису на два види.
3. Оціните ефективність моделі.

Хід роботи.

Для виконання завдання будемо використовувати файл з даними, який доступный за посиланням: <https://www.kaggle.com/datasets/muratkokludataset/rice-dataset-commeo-and-osmancik>

В ньому розташовані дані про 2 види рису **Commeo and Osmancik**

Перегляньте, яка інформація там є:

**Area: повертає кількість пікселів у межах рисового зерна.**

**Perimeter: обчислює окружність шляхом обчислення відстані між пікселями навколо меж рисового зерна.**

**Major Axis Length: це найдовша лінія, яку можна накреслити на рисовому зерні, тобто відстань головної осі.**

**Minor Axis Length: дає найкоротша лінія, яку можна намалювати на рисовому зерні, тобто мала відстань по осі.**

**Eccentricity: він вимірює, наскільки округлим є еліпс, який має ті самі моменти, що й рисове зерно.**

**Convex Area: повертає кількість пікселів найменшої опуклої оболонки області, утвореної рисовим зерном.**

**Extent: повертає співвідношення області, утвореної рисовим зерном, до пікселів обмежувальної рамки**

**Клас: Коммео і Османчик**

1 крок. Створіть новий jupyter notebook.

2. Імпортуйте необхідні бібліотеки



3. Завантажте Ваші данні:

rice\_dataset\_raw = pd.read\_csv("https://download.mlcc.google.com/mledu-datasets/Rice\_Cammeo\_Osmancik.csv")

1. Завантажте ці стовбчики з набру даних Ось пояснення для кожного з показників у наборі даних:
2. **Area**: Площа зерна рису в пікселях. Відображає розмір зерна.
3. **Perimeter**: Периметр зерна рису в пікселях. Вимірює обхід зерна.
4. **Major\_Axis\_Length**: Довжина головної осі еліпса, що описує зерно. Це найбільша відстань між двома точками на краю зерна.
5. **Minor\_Axis\_Length**: Довжина малої осі еліпса, що описує зерно. Це найменша відстань між двома точками на краю зерна.
6. **Eccentricity**: Еccentricity (еліптичність) — відношення між відстанями фокусів еліпса до довжини його головної осі. Вимірює, наскільки зерно є витягнутим.
7. **Convex\_Area**: Площа опуклої оболонки зерна. Це площа, яку займає зерно, якщо його «упакувати» в опуклу форму.
8. **Extent**: Відношення площі зерна до площі його обгортки. Вимірює, наскільки зерно заповнює свою опуклу оболонку.
9. **Class**: Клас або сорт зерна рису (наприклад, Osmancik або Cammeo). Це цільова змінна для класифікації.

rice\_dataset = rice\_dataset\_raw[[

 'Area',

 'Perimeter',

 'Major\_Axis\_Length',

 'Minor\_Axis\_Length',

 'Eccentricity',

 'Convex\_Area',

 'Extent',

 'Class',

]]

1. Побудуйте діаграми, які дозволяють візуалізувати два класи рису в цих даних:

for x\_axis\_data, y\_axis\_data in [

 ('Area', 'Eccentricity'),

 ('Convex\_Area', 'Perimeter'),

 ('Major\_Axis\_Length', 'Minor\_Axis\_Length'),

 ('Perimeter', 'Extent'),

 ('Eccentricity', 'Major\_Axis\_Length'),

]:

 px.scatter(rice\_dataset, x=x\_axis\_data, y=y\_axis\_data, color='Class').show()

Пари змінних, які ви використовуєте для візуалізації за допомогою plotly.express.scatter(),вибрані для аналізу кореляцій або закономірностей у даних. Давайте розглянемо логіку вибору цих пар:

1. **('Area', 'Eccentricity')**
	* Area (площа об'єкта) показує загальний розмір рисового зерна.
	* Eccentricity (ексцентриситет) описує, наскільки об'єкт витягнутий (наближений до еліпса).
	* Ця пара може допомогти зрозуміти, чи великі зерна зазвичай більш видовжені або округлі.
2. **('Convex\_Area', 'Perimeter')**
	* Convex\_Area — площа опуклої оболонки (мінімального опуклого багатокутника, що містить об'єкт).
	* Perimeter — довжина контуру об'єкта.
	* Дослідження цієї пари може показати, наскільки об'єкт має вигини або виступи (наприклад, якщо Perimeter значно більший за Convex\_Area, то форма може бути складною).
3. **('Major\_Axis\_Length', 'Minor\_Axis\_Length')**
	* Major\_Axis\_Length — довжина головної осі еліпса, який апроксимує об'єкт.
	* Minor\_Axis\_Length — довжина меншої осі.
	* Очікується кореляція між цими характеристиками, оскільки довші зерна матимуть більші значення обох осей.
4. **('Perimeter', 'Extent')**
	* Perimeter — довжина межі об'єкта.
	* Extent — відношення площі об'єкта до площі його охоплюючого прямокутника.
	* Ця пара може показати, як форма об'єкта впливає на те, наскільки він заповнює свою мінімальну охоплюючу область.
5. **('Eccentricity', 'Major\_Axis\_Length')**
	* Eccentricity показує, наскільки об'єкт витягнутий.
	* Major\_Axis\_Length є прямою мірою довжини цього витягування.
	* Ця пара допомагає оцінити, як ексцентриситет змінюється зі збільшенням довжини головної осі.





І т.п. Ми можемо бачити, що за всіми ціми показниками виявляються чітки залежности виду рису.

Побудуємо різні моделі бінарної класифікації та порівняємо їх якість:

# 1. Підготуйте ознаки та мітки

label\_column = 'Class'

features = rice\_dataset[['Area', 'Perimeter', 'Major\_Axis\_Length', 'Minor\_Axis\_Length', 'Eccentricity', 'Convex\_Area', 'Extent']]

labels = rice\_dataset[label\_column].map({'Cammeo': 1, 'Osmancik': 0}).to\_numpy() # Призначте мітки

# 2. Розділіть дані на навчальний, валідаційний та тестовий набори



from sklearn.model\_selection import train\_test\_split

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# 3. Нормалізуйте дані

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_val\_scaled = scaler.transform(X\_val)

X\_test\_scaled = scaler.transform(X\_test)

Спочатку побудуємо дві класичних моделі класифікації – на основі методу опорних векторів та на основі дерев рішень

#4. Модель на основі методу опорних векторів.

# Створення та навчання моделі SVM

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

model = SVC(kernel='linear') # Використовуємо лінійне ядро

model.fit(X\_train\_scaled, y\_train)

# Прогнозування на тестових даних

y\_pred = model.predict(X\_test\_scaled)

# Оцінка точності моделі

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Точність моделі SVM: {accuracy \* 100:.2f}%')

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))



1. **Accuracy (Точність)** – **92.39%**
	* Це частка правильно класифікованих прикладів.
	* Усього 381 тестовий приклад → **SVM передбачив правильно 92.39% з них**.
2. **Precision (Точність для класів)**
	* Для класу 0: **92%** → З усіх передбачених як 0, **92% дійсно належать до 0**.
	* Для класу 1: **93%** → З усіх передбачених як 1, **93% дійсно належать до 1**.
3. **Recall (Повнота)**
	* Для класу 0: **95%** → З усіх реальних 0, **95% правильно передбачені**.
	* Для класу 1: **90%** → З усіх реальних 1, **90% правильно передбачені**.
4. **F1-score (Середнє між Precision і Recall)**
	* **Для класу 0: 93%**
	* **Для класу 1: 92%**
	* Загалом **баланс precision і recall хороший**.

Якщо важливо зменшити кількість пропущених 1, можна:

* **Змінити поріг (threshold) класифікації**.
* **Спробувати інший SVM-ядро** (rbf, poly).

# 5. Модель на основі дерев рішень

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

# Створення та навчання моделі на основі дерев рішень

model = DecisionTreeClassifier(criterion='entropy', max\_depth=5, min\_samples\_split=3, random\_state=42)

model.fit(X\_train\_scaled, y\_train)

# Прогнозування на тестових даних

y\_pred = model.predict(X\_test\_scaled)

# Оцінка точності моделі

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Точність моделі TreeDecisions: {accuracy \* 100:.2f}%')

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))



Зробіть висновки.

#6 Спробуємо застосувати модель на основі нейронної мережі.

import tensorflow as tf

model = tf.keras.Sequential([

 tf.keras.Input(shape=(X\_train\_scaled.shape[1],)),

 tf.keras.layers.Dense(16, activation='relu'),

 tf.keras.layers.Dense(8, activation='relu'),

 tf.keras.layers.Dense(1, activation='sigmoid') # Вихідний шар для бінарної класифікації

])

Цей код створює модель нейронної мережі для задачі бінарної класифікації за допомогою Keras (в рамках TensorFlow). Ось детальний опис:

1. **tf.keras.Sequential([...])**: Використовується для створення послідовної моделі, де шари йдуть один за одним.
2. **tf.keras.Input(shape=(X\_train\_scaled.shape[1],))**: Це вхідний шар, де вказується форма вхідних даних. У даному випадку, розмір входу буде рівним кількості ознак у X\_train\_scaled.
3. **tf.keras.layers.Dense(16, activation='relu')**: Це перший прихований шар, що містить 16 нейронів. Кожен нейрон використовує функцію активації ReLU ('relu'), яка перетворює вхідні значення для нелінійної обробки.
4. **tf.keras.layers.Dense(8, activation='relu')**: Другий прихований шар містить 8 нейронів і також використовує функцію активації ReLU.
5. **tf.keras.layers.Dense(1, activation='sigmoid')**: Це вихідний шар. Він містить 1 нейрон із функцією активації 'sigmoid'. Використовується для бінарної класифікації, оскільки функція активації sigmoid перетворює вихід у діапазон [0, 1], що інтерпретується як ймовірність приналежності до одного з двох класів.

**ReLU (Rectified Linear Unit)** — це активаційна функція, яка широко використовується в нейронних мережах, особливо в прихованих шарах. Вона має таку формулу:



# 7. Скомпонуйте модель

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# 8. Навчіть модель

history = model.fit(X\_train\_scaled, y\_train, validation\_data=(X\_val\_scaled, y\_val), epochs=50, batch\_size=16)

# 9. Оцініть модель на тестовому наборі

test\_loss, test\_accuracy = model.evaluate(X\_test\_scaled, y\_test)

print(f'Test accuracy: {test\_accuracy:.4f}')

# 9. Виконайте прогнозування на тестовому наборі

test\_predictions = model.predict(X\_test\_scaled)

test\_classes = (test\_predictions > 0.5).astype(int)

# 10. Оцініть ефективність на тестовому наборі

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, test\_classes))

Ви отримаєте наступний звіт: #**11. Спробуйте запустити модель з іншим порогом:**

test\_predictions = model.predict(X\_test\_scaled)

test\_classes = (test\_predictions > 0.7).astype(int)

**Знову оцініть результати та зробіть висновки.**

Самостійно: оберіть найкращу модель визначення виду рису, додаючи чи прибираючі ознаки з набору даних. Проаналізуйте, чи збільшується якість моделі при зміні ознак.