Прізвище: **КИРИЛЮК**
Ім'я: **Дмитро**
Група: **ПП-22**
Варіант: **08**
Дата захисту: **21.04.2025р**.

Кафедра: **САПР**
Дисципліна: **Системи інтелектуального аналізу та візуалізації даних**
Перевірив: **Андрій КЕРНИЦЬКИЙ**

ЗВІТ

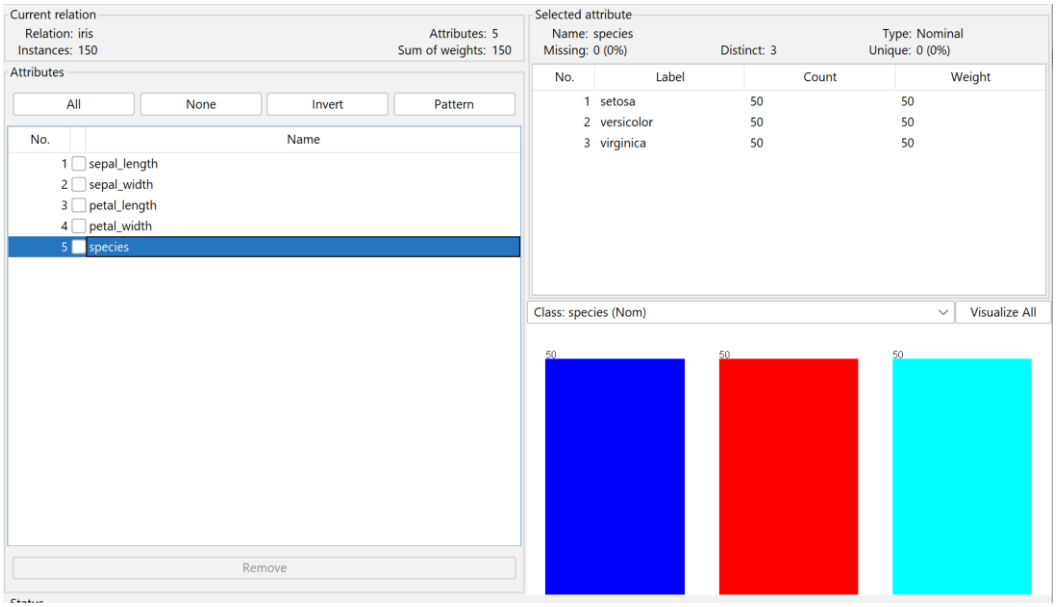
до лабораторної роботи №11
на тему **“Класифікація методом опорних векторів.”**

Мета роботи: Ознайомитися та отримати навички побудови моделей класифікації за допомогою Data Mining GUI бібліотеки Weka. На практиці вивчити роботу методу опорних векторів, навчитися інтерпретувати результати роботу класифікатора.

Індивідуальне завдання:

1. Для індивідуального завдання вирішіть задачу класифікації за допомогою методу опорних векторів (functions.SMO).
2. Змінюючи параметри налаштування алгоритму, спробуйте досягти найвищої якості навчання класифікатора.
3. Здійсніть класифікацію методом опорних векторів із датасетом використаним у WEKA у Excel за допомогою пакету Excel2SVM (https://www.bioinformatics.org/Excel2SVM/)
4. Порівняйте отримані результати від різних систем.
5. У звіті надайте результати роботи алгоритму, його налаштування, а також результати порівняння.

1 частина:



Щоб покращити якість класифікації за допомогою методу опорних векторів (SMO) в Weka, я можу змінювати наступні параметри:

1. **C (Regularization Parameter)** – контролює баланс між простотою моделі та її здатністю правильно класифікувати навчальні дані.
    - Вищі значення (наприклад, 10, 100) можуть покращити навчання, але підвищують ризик перенавчання.
    - Нижчі значення (наприклад, 0.1, 0.01) можуть зробити модель більш узагальненою.
2. **Kernel (Ядро)** – визначає, як дані трансформуються в багатовимірний простір.
    - PolyKernel (Поліноміальне ядро)
        - Важливий параметр E (Exponent, степінь полінома) – чим вище значення, тим складніші межі розділу.
    - RBFKernel (радіально-базисне ядро)
        - Параметр γ (gamma) визначає, наскільки далеко вплив однієї точки поширюється на інші. Менші значення роблять модель більш гнучкою.
    - PukKernel та StringKernel – рідше використовуються.
3. **Tolerance Parameter** – визначає точність рішення для оптимізації.
    - Менші значення (0.0001, 0.00001) можуть покращити точність, але збільшать час навчання.
4. **Epsilon** – використовується в ε-SVR (Support Vector Regression).
    - Вищі значення (1E-8, 1E-12) дають більшу гнучкість у допуску похибки.
5. **FilterType (Тип нормалізації)** –
    - Normalize training data – рекомендується для SVM, якщо дані мають різний масштаб.
    - Можна спробувати "Standardize training data".
6. **Kernel Cache Size (C у PolyKernel)** –
    - Збільшення кешу (наприклад, 500000) може прискорити навчання, якщо є достатньо пам'яті.
7. **Random Seed** –
    - Впливає на випадковість розбиття даних. Спробуйте змінити (наприклад, 42, 100, 1234), щоб оцінити стабільність результатів.
8. **Крос-валідація (numFolds)** –
    - Якщо numFolds = -1, Weka використовує стандартне розділення даних.
    - Встановлення numFolds = 5 або 10 може покращити узагальнення.

Одним із ключових параметрів алгоритму є вибір ядра, яке визначає, як саме модель розділяє простір ознак, відносно нього і будемо керуватись.

Для оцінки впливу різних ядер на якість класифікації були випробувані три варіанти:

1. **Radial Basis Function Kernel (RBF)** – радіальна базисна функція
2. **Normalized Polynomial Kernel** – нормалізоване поліноміальне ядро
3. **Polynomial Kernel** – стандартне поліноміальне ядро

RBF Kernel

```
=== Summary ===

Correctly Classified Instances         139               92.6667 %
Incorrectly Classified Instances        11                7.3333 %
Kappa statistic                          0.89
Mean absolute error                      0.2385
Root mean squared error                  0.3006
Relative absolute error                 53.6667 %
Root relative squared error             63.7704 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     setos
                 0,920    0,070    0,868      0,920    0,893      0,838    0,925     0,825     versi
                 0,860    0,040    0,915      0,860    0,887      0,833    0,957     0,871     virgi
Weighted Avg.    0,927    0,037    0,928      0,927    0,927      0,891    0,961     0,899

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 46  4 |  b = versicolor
  0  7 43 |  c = virginica
```

Normalized Poly Kernel

```
=== Summary ===

Correctly Classified Instances         102               68      %
Incorrectly Classified Instances        48               32      %
Kappa statistic                          0.52
Mean absolute error                      0.2933
Root mean squared error                  0.381
Relative absolute error                 66      %
Root relative squared error             80.829  %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     setos
                 0,980    0,470    0,510      0,980    0,671      0,501    0,755     0,507     versi
                 0,060    0,010    0,750      0,060    0,111      0,146    0,760     0,515     virgi
Weighted Avg.    0,680    0,160    0,753      0,680    0,594      0,549    0,838     0,674

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 49  1 |  b = versicolor
  0 47  3 |  c = virginica
```

Poly Kernel

```
=== Summary ===

Correctly Classified Instances         145               96.6667 %
Incorrectly Classified Instances         5                3.3333 %
Kappa statistic                          0.95
Mean absolute error                      0.2296
Root mean squared error                  0.2854
Relative absolute error                 51.6667 %
Root relative squared error             60.553  %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     setos
                 0,980    0,040    0,925      0,980    0,951      0,927    0,970     0,913     versi
                 0,920    0,010    0,979      0,920    0,948      0,925    0,975     0,940     virgi
Weighted Avg.    0,967    0,017    0,968      0,967    0,967      0,951    0,982     0,951

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 49  1 |  b = versicolor
  0  4 46 |  c = virginica
```

Після тестування різних варіантів було встановлено, що найкращу якість класифікації (96,7% правильно класифікованих прикладів) забезпечує Poly Kernel. Ця модель майже ідеально розрізняє класи.

2 частина:

Я класифікував дані через програмний модуль мови Python sklearn використовуючи ядро для алгоритму SVM Poly Kernel.

Програмна реалізація:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# === 1. Завантаження даних ===
file_path = './iris.csv'  # заміни на свій файл
df = pd.read_csv(file_path)

# === 2. Розділення ознак і цільової змінної ===
X = df.iloc[:, :-1]               # усі стовпці, крім останнього — ознаки
y_raw = df.iloc[:, -1]            # останній стовпець — текстові класи

# === 3. Кодування класів у числа ===
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y_raw)

# === 4. Розбиття на train/test ===
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# === 5. Класифікація SVM з poly kernel ===
clf = SVC(kernel='poly', degree=3, C=1.0)  # можна змінити degree або C для експериментів
clf.fit(X_train, y_train)

# === 6. Прогнозування і оцінка ===
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Результати виконання програми:

```
Accuracy: 0.9777777777777777

Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      0.92      0.96        13
   virginica       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45


Confusion Matrix:
 [[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
```

3 частина :

При порівнянні результатів класифікації методом опорних векторів (SVM), отриманих у різних системах (Weka та Python (sklearn)), можна зробити наступні висновки:

Спільні особливості:

1. Поліноміальне ядро (Poly Kernel) дало найкращі результати класифікації в обох системах. Це свідчить про його ефективність для даного набору даних.
2. Висока точність класифікації: точність перевищила 96% як у WEKA (96.67%), так і в sklearn (97.78%) при відповідному налаштуванні параметрів моделі.
3. Класи розпізнані зі схожою ефективністю — зокрема, клас *setosa* було ідентифіковано без помилок в обох випадках. Це говорить про схожість виявлених моделей для класифікації.
4. Підтримуючі вектори (support vectors) виявили подібні закономірності у даних: найкраще розділяються *setosa*, а складніше — *versicolor* та *virginica*.

Відмінності:

1. Реалізація алгоритму в обох системах відрізняється:
   - У WEKA використовувався класичний модуль functions.SMO.
   - У scikit-learn застосовано стратегію One-vs-Rest (OvR), де кожен клас порівнюється з усіма іншими.
2. У WEKA результати класифікації оцінювались через повну вибірку, а у sklearn — шляхом поділу на навчальну і тестову частину (напр., test_size=0.3), що впливає на точність порівняння.
3. Кількість support vectors у реалізації sklearn доступна як .support_ та може бути проаналізована додатково, а в WEKA її кількість виводиться неявно.
4. У WEKA автоматично розраховуються додаткові метрики як Mean Absolute Error, Kappa Statistic, що не входять у стандартний звіт sklearn.

Обидві системи — WEKA та scikit-learn — продемонстрували високу ефективність при класифікації даних методом опорних векторів.
Поліноміальне ядро підтвердило свою придатність для роботи з даними типу Iris, що мають чітку структурованість.
Scikit-learn забезпечує більшу гнучкість у налаштуванні моделей та обробці результатів програмно, тоді як WEKA надає зручний інтерфейс і швидке налаштування через GUI.
Отже, при правильному виборі гіперпараметрів, метод опорних векторів з поліноміальним ядром гарантує високу якість класифікації незалежно від обраної платформи.

Висновок: У результаті виконання лабораторної роботи було успішно застосовано метод опорних векторів (SVM) для класифікації квітів ірису за їхніми морфологічними ознаками. Дослідження різних типів ядер показало, що поліноміальне ядро забезпечує найвищу точність класифікації (до 97.78%) порівняно з іншими варіантами. Порівняння результатів у системах WEKA та модуля scikit-learn у Python підтвердило високу ефективність і надійність методу SVM за умови правильного налаштування параметрів. Це свідчить про чітке розділення класів (*setosa*, *versicolor*, *virginica*) у просторі ознак, що робить даний підхід доцільним для розв'язання задач біологічної класифікації та аналізу подібних структурованих даних.