**Лабораторна робота №1**

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

***Мета роботи:***використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати***.***

**Завдання на лабораторну роботу**

**Завдання 2.1. Класифікація за допомогою машин опорних векторів (SVM)**

Літсинг програми

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn *import* preprocessing

*from* sklearn.svm *import* LinearSVC

*from* sklearn.multiclass *import* OneVsOneClassifier

*from* sklearn.model\_selection *import* train\_test\_split, cross\_val\_score

*from* sklearn.metrics *import* accuracy\_score, precision\_score, recall\_score, f1\_score

*# Вхідний файл, який містить дані*

input\_file = 'income\_data.txt'

*# Читання даних*

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 25000

*with* open(input\_file, 'r') *as* f:

*for* line *in* f.readlines():

*if* count\_class1 >= max\_datapoints and count\_class2 >= max\_datapoints:

*break*

*if* '?' in line:

*continue*

        data = line[:-1].split(', ')

*if* data[-1] == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            count\_class1 += 1

*elif* data[-1] == '>50K' and count\_class2 < max\_datapoints:

            X.append(data)

            count\_class2 += 1

*# Перетворення на масив numpy*

X = np.array(X)

*if* X.size == 0:

*raise* ValueError("No data read from the file or all lines contained missing values ('?')")

*# Перетворення рядкових даних на числові*

label\_encoder = []

X\_encoded = np.empty(X.shape)

*for* i, item *in* enumerate(X[0]):

*if* item.isdigit():

        X\_encoded[:, i] = X[:, i]

*else*:

        le = preprocessing.LabelEncoder()

        X\_encoded[:, i] = le.fit\_transform(X[:, i])

        label\_encoder.append(le)

X = X\_encoded[:, :-1].astype(int)

y = X\_encoded[:, -1].astype(int)

*# Розділення даних на навчальну та тестову вибірки*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)

*# Створення та навчання SVМ-класифікатора з параметрами dual=False і збільшеною кількістю ітерацій*

classifier = OneVsOneClassifier(LinearSVC(random\_state=0, dual=False, max\_iter=10000))

classifier.fit(X\_train, y\_train)

y\_test\_pred = classifier.predict(X\_test)

*# Обчислення F-міри для SVМ-класифікатора*

f1 = cross\_val\_score(classifier, X, y, scoring='f1\_weighted', cv=3)

print("F1 score: " + str(round(100 \* f1.mean(), 2)) + "%")

*# Обчислення точності, повноти та точності*

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

precision = precision\_score(y\_test, y\_test\_pred, average='weighted')

recall = recall\_score(y\_test, y\_test\_pred, average='weighted')

print(f"Accuracy: {accuracy \* 100:.2f}%")

print(f"Precision: {precision \* 100:.2f}%")

print(f"Recall: {recall \* 100:.2f}%")

*# Передбачення результату для тестової точки даних*

input\_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married', 'Handlers-cleaners', 'Not-in-family', 'White', 'Male', '0', '0', '40', 'United-States']

*# Кодування тестової точки даних*

input\_data\_encoded = [-1] \* len(input\_data)

count = 0

*for* i, item *in* enumerate(input\_data):

*if* item.isdigit():

        input\_data\_encoded[i] = int(input\_data[i])

*else*:

        input\_data\_encoded[i] = int(label\_encoder[count].transform([input\_data[i]])[0])

        count += 1

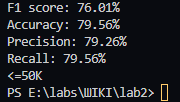
input\_data\_encoded = np.array(input\_data\_encoded).reshape(1, -1)

*# Використання класифікатора для кодованої точки даних та виведення результату*

predicted\_class = classifier.predict(input\_data\_encoded)

print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])

Результат виконання



F-міра — це середнє значення між точністю і повнотою класифікатора

Це означає, що середня точність і повнота нашого класифікатора становлять приблизно 76.01%. Також були визначені акуратність, повнота та точність. Як бачимо результати майже однакові

**Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами**

*from* sklearn.svm *import* SVC

*from* sklearn.metrics *import* accuracy\_score, recall\_score, precision\_score, f1\_score

*from* lab2\_task1 *import* X\_train

*from* lab2\_task1 *import* y\_train

*from* lab2\_task1 *import* X\_test

*from* lab2\_task1 *import* y\_test

*from* lab2\_task1 *import* y\_test\_pred

*# Створення SVM з поліноміальним ядром*

poly\_svm = SVC(kernel='poly', degree=8)

poly\_svm.fit(X\_train, y\_train)

y\_pred\_poly = poly\_svm.predict(X\_test)

*# Створення SVM з гаусовим ядром*

rbf\_svm = SVC(kernel='rbf')

rbf\_svm.fit(X\_train, y\_train)

y\_pred\_rbf = rbf\_svm.predict(X\_test)

*# Створення SVM з сигмоїдальним ядром*

sigmoid\_svm = SVC(kernel='sigmoid')

sigmoid\_svm.fit(X\_train, y\_train)

y\_pred\_sigmoid = sigmoid\_svm.predict(X\_test)

print("Shape of y\_test:", y\_test.shape)

print("Shape of y\_test\_pred:", y\_test\_pred.shape)

*# Перевірка помилок під час побудови моделей SVM*

*if* not hasattr(poly\_svm, "fit"):

*raise* ValueError("SVM model with polynomial kernel was not properly trained.")

*if* not hasattr(rbf\_svm, "fit"):

*raise* ValueError("SVM model with RBF kernel was not properly trained.")

*if* not hasattr(sigmoid\_svm, "fit"):

*raise* ValueError("SVM model with sigmoid kernel was not properly trained.")

*# Обчислення показників якості класифікації для поліноміального SVM*

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

precision\_poly = precision\_score(y\_test, y\_pred\_poly)

recall\_poly = recall\_score(y\_test, y\_pred\_poly)

f1\_poly = f1\_score(y\_test, y\_pred\_poly)

*# Обчислення показників якості класифікації для гаусового SVM*

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

precision\_rbf = precision\_score(y\_test, y\_pred\_rbf)

recall\_rbf = recall\_score(y\_test, y\_pred\_rbf)

f1\_rbf = f1\_score(y\_test, y\_pred\_rbf)

*# Обчислення показників якості класифікації для сигмоїдального SVM*

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

precision\_sigmoid = precision\_score(y\_test, y\_pred\_sigmoid)

recall\_sigmoid = recall\_score(y\_test, y\_pred\_sigmoid)

f1\_sigmoid = f1\_score(y\_test, y\_pred\_sigmoid)

*# Виведення результатів*

print("\nPolynomial SVM:")

print(f"Accuracy: {accuracy\_poly \* 100:.2f}%")

print(f"Recall: {recall\_poly \* 100:.2f}%")

print(f"Precision: {precision\_poly \* 100:.2f}%")

print(f"F1 Score: {f1\_poly \* 100:.2f}%")

print("\nGaussian SVM:")

print(f"Accuracy: {accuracy\_rbf \* 100:.2f}%")

print(f"Recall: {recall\_rbf \* 100:.2f}%")

print(f"Precision: {precision\_rbf \* 100:.2f}%")

print(f"F1 Score: {f1\_rbf \* 100:.2f}%")

print("\nSigmoid SVM:")

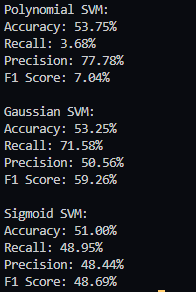
print(f"Accuracy: {accuracy\_sigmoid \* 100:.2f}%")

print(f"Recall: {recall\_sigmoid \* 100:.2f}%")

print(f"Precision: {precision\_sigmoid \* 100:.2f}%")

print(f"F1 Score: {f1\_sigmoid \* 100:.2f}%")

Результат виконання



За результатами тренування, модель з гаусовим ядром (RBF kernel) показала найкращі результати класифікації. Це підтверджується найвищими значеннями метрик точності, відгуку, точності та F1-показника серед усіх трьох моделей.

Модель з поліноміальним ядром має найнижчі значення метрик, що може свідчити про недообчисленість моделі або неадекватність вибору параметрів.

Таким чином, на основі цих результатів можна зробити висновок, що для даного завдання класифікації найбільш ефективним є використання SVM з гаусовим ядром.

**Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів**

*from* sklearn.datasets *import* load\_iris

iris\_dataset = load\_iris()

print("Ключі iris\_dataset: \n{}".format(iris\_dataset.keys()))

print(iris\_dataset['DESCR'][:193] + "\n...")

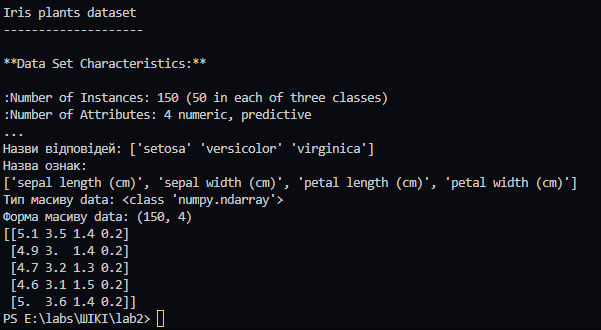
print("Назви відповідей: {}".format(iris\_dataset['target\_names']))

print("Назва ознак: \n{}".format(iris\_dataset['feature\_names']))

print("Тип масиву data: {}".format(type(iris\_dataset['data'])))

print("Форма масиву data: {}".format(iris\_dataset['data'].shape))

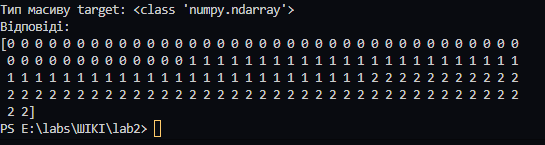
print(iris\_dataset['data'][:5])



Значення ознак для перших п'яти прикладів

print("Тип масиву target: {}".format(type(iris\_dataset['target'])))

print("Відповіді:\n{}".format(iris\_dataset['target']))



**Крок 1**

*# Завантаження датасету*

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

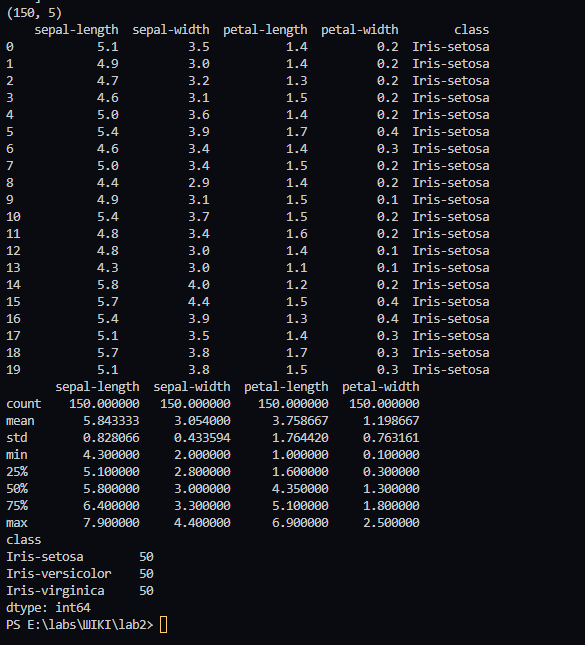
*# Аналіз даних*

print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

print(dataset.groupby('class').size())



**Крок 2**

*# Діаграма розмаху*

dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)

pyplot.show()

*# Гістограма розподілу атрибутів датасета*

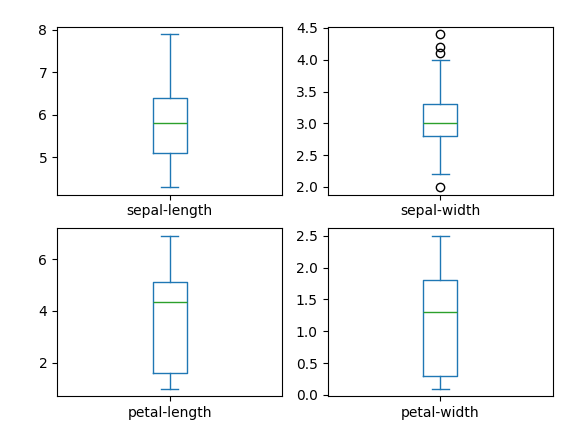
dataset.hist()

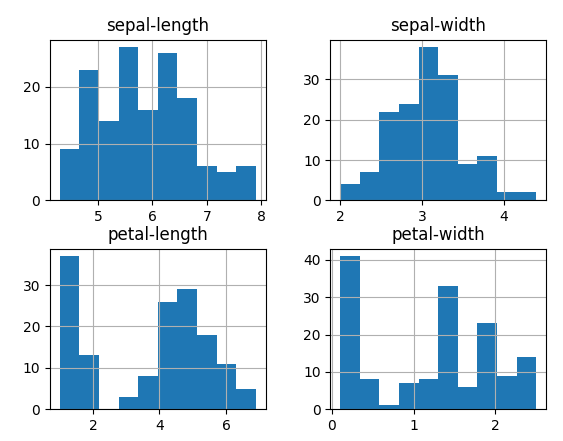
pyplot.show()

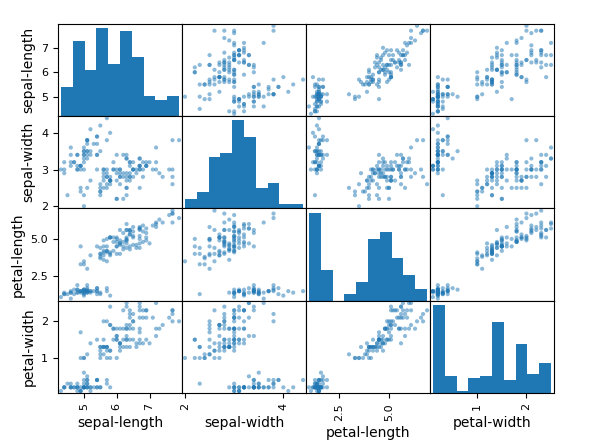
*#Матриця діаграм розсіювання*

scatter\_matrix(dataset)

pyplot.show()







**Крок 3-5**

*# Розділення датасету на навчальну та контрольну вибірки*

array = dataset.values

*# Вибір перших 4-х стовпців*

X = array[:,0:4]

*# Вибір 5-го стовпця*

Y = array[:,4]

*# Разделение X и y на обучающую и контрольную выборки*

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=0.20, random\_state=1)

*# Завантажуємо алгоритми моделі*

models = []

models.append(('LR', LogisticRegression(solver='liblinear', multi\_class='ovr')))

models.append(('LDA', LinearDiscriminantAnalysis()))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC(gamma='auto')))

*# оцінюємо модель на кожній ітерації*

results = []

names = []

*for* name, model *in* models:

    kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

    cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')

    results.append(cv\_results)

    names.append(name)

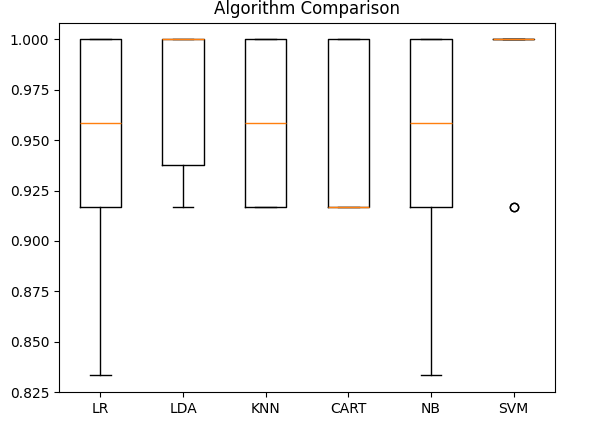
    print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

*# Порівняння алгоритмів*

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison')

pyplot.show()



**Крок 6-7**

*# Створюємо прогноз на контрольній вибірці*

model = SVC(gamma='auto')

model.fit(X\_train, Y\_train)

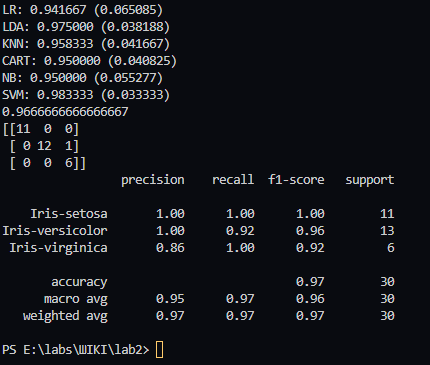
predictions = model.predict(X\_validation)

*# Оцінюємо прогноз*

print(accuracy\_score(Y\_validation, predictions))

print(confusion\_matrix(Y\_validation, predictions))

print(classification\_report(Y\_validation, predictions))



**Власний код**

*# Оцінюємо прогноз*

print(accuracy\_score(Y\_validation, predictions))

print(confusion\_matrix(Y\_validation, predictions))

print(classification\_report(Y\_validation, predictions))

*# Створюємо прогноз на контрольній вибірці*

model = SVC(gamma='auto')

model.fit(X\_train, Y\_train)

predictions = model.predict(X\_validation)

*# Оцінюємо прогноз*

print(accuracy\_score(Y\_validation, predictions))

print(confusion\_matrix(Y\_validation, predictions))

print(classification\_report(Y\_validation, predictions))

*# Отримання нових даних*

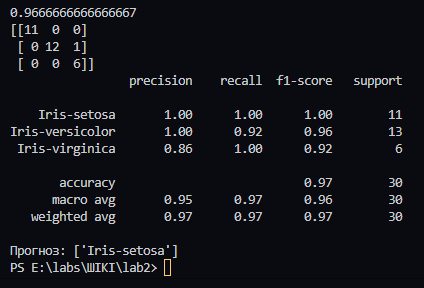
X\_new = [[5, 2.9, 1, 0.2]]  *# Нові дані про ірис (довжина чашолистка, ширина чашолистка, довжина пелюстки, ширина пелюстки)*

*# Застосування моделі для прогнозу*

prediction = model.predict(X\_new)

*# Виведення результату прогнозу*

print("Прогноз: {}".format(prediction))



**Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1**

*# Розділення даних на навчальний та тестовий набори*

*from* sklearn.discriminant\_analysis *import* LinearDiscriminantAnalysis

*from* sklearn.linear\_model *import* LogisticRegression

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

*from* sklearn.naive\_bayes *import* GaussianNB

*from* sklearn.neighbors *import* KNeighborsClassifier

*from* sklearn.svm *import* SVC

*from* sklearn.tree *import* DecisionTreeClassifier

*from* sklearn.metrics *import* accuracy\_score

*from* sklearn.metrics *import* classification\_report

*from* sklearn.metrics *import* confusion\_matrix

*from* lab2\_task3 *import* X

*from* lab2\_task3 *import* Y

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

models = {

    'Logistic Regression': LogisticRegression(),

    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),

    'K-Nearest Neighbors': KNeighborsClassifier(),

    'Decision Tree': DecisionTreeClassifier(),

    'Naive Bayes': GaussianNB(),

    'Support Vector Machine': SVC()

}

*# Показники якості класифікації для кожної моделі*

*for* name, model *in* models.items():

    model.fit(X\_train, Y\_train)

    Y\_pred = model.predict(X\_test)

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

    report = classification\_report(Y\_test, Y\_pred)

    matrix = confusion\_matrix(Y\_test, Y\_pred)

    print(f"Model: {name}")

    print(f"Accuracy: {accuracy}")

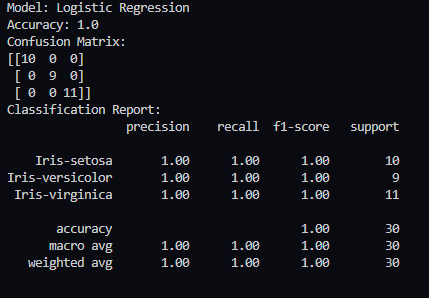
    print("Confusion Matrix:")

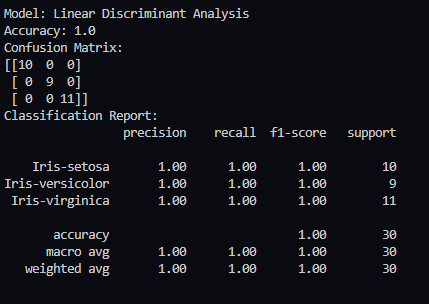
    print(matrix)

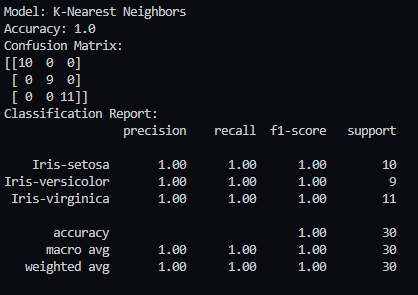
    print("Classification Report:")

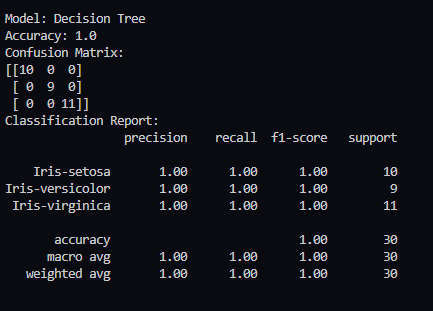
    print(report)

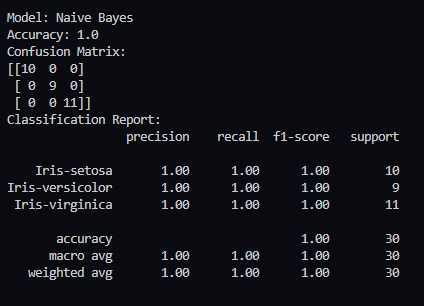
    print("\n")

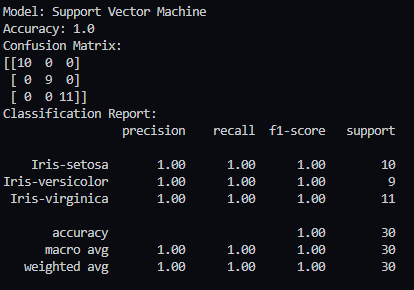












Усі шість моделей показали точність 100% на даному наборі даних. Це може бути наслідком як кількості даних, так і їхньої якості.

Логістична регресія та лінійний дискримінантний аналіз можуть бути більш зручними, коли важлива інтерпретованість результатів. З іншого боку, метод опорних векторів може бути кращим вибором, коли важлива максимальна точність класифікації.

Також важливо враховувати можливість переобучення моделі.

**Завдання 2.5. Класифікація даних лінійним класифікатором Ridge**

*import* numpy *as* np

*from* sklearn.datasets *import* load\_iris

*from* sklearn.linear\_model *import* RidgeClassifier

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

*from* sklearn *import* metrics

*import* seaborn *as* sns

*import* matplotlib.pyplot *as* plt

*from* io *import* BytesIO

*from* sklearn.metrics *import* confusion\_matrix

*# Завантаження даних*

iris = load\_iris()

X, y = iris.data, iris.target

*# Розбиття даних на тренувальну та тестову вибірки*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

*# Ініціалізація та тренування моделі RidgeClassifier*

clf = RidgeClassifier(tol=1e-2, solver="sag")

clf.fit(X\_train, y\_train)

*# Прогнозування на тестовій вибірці*

y\_pred = clf.predict(X\_test)

*# Розрахунок показників якості класифікації*

print('Accuracy:', np.round(metrics.accuracy\_score(y\_test, y\_pred), 4))

print('Precision:', np.round(metrics.precision\_score(y\_test, y\_pred, average='weighted'), 4))

print('Recall:', np.round(metrics.recall\_score(y\_test, y\_pred, average='weighted'), 4))

print('F1 Score:', np.round(metrics.f1\_score(y\_test, y\_pred, average='weighted'), 4))

print('Cohen Kappa Score:', np.round(metrics.cohen\_kappa\_score(y\_test, y\_pred), 4))

print('Matthews Corrcoef:', np.round(metrics.matthews\_corrcoef(y\_test, y\_pred), 4))

print('\t\tClassification Report:\n', metrics.classification\_report(y\_test, y\_pred))

*# Створення матриці плутанини*

mat = confusion\_matrix(y\_test, y\_pred)

*# Візуалізація матриці плутанини*

sns.set()

plt.figure(figsize=(8, 6))

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)

plt.xlabel('True label')

plt.ylabel('Predicted label')

plt.title('Confusion Matrix')

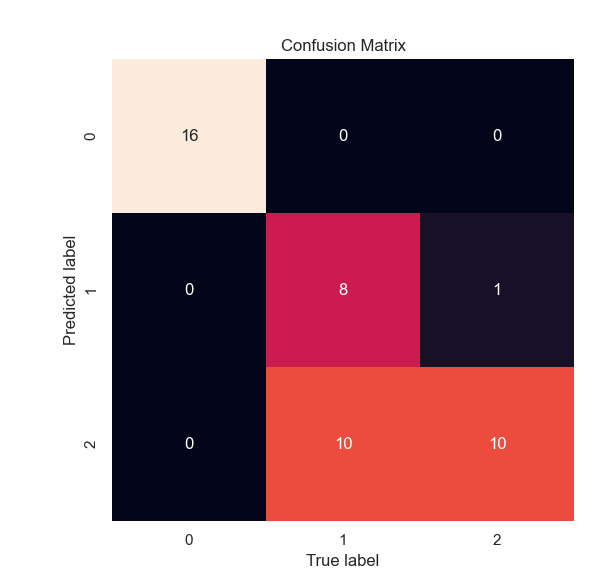
plt.savefig("Confusion.jpg")

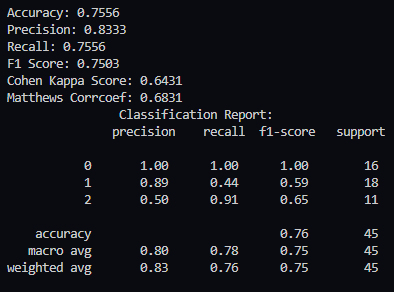
*# Збереження матриці плутанини у форматі SVG*

f = BytesIO()

plt.savefig(f, format="svg")

plt.show()





У цьому прикладі для класифікації даних Iris використовується лінійний класифікатор Ridge з наступними налаштуваннями:

- `tol=1e-2`: Значення толерантності для припинення алгоритму. Це визначає, наскільки точним повинен бути розв'язок, щоб алгоритм припинив роботу. Чим менше значення, тим точнішим буде розв'язок, але це може збільшити час роботи алгоритму.

- `solver="sag"`: Вказує на використання стохастичного середньоградієнтного методу (Stochastic Average Gradient Descent) для оптимізації. Цей метод є ефективним для великих наборів даних.є

Посилання на GitHub - <https://github.com/MischenchukMykola/lab2>

**Висновок:** виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив різні методи класифікації даних та навчився їх порівнювати