Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем і технологій

# Лабораторна робота №4

## **Прикладні задачі машинного навчання**

“Класифікація методом k найближчих сусідів і набір даних Digits, частина 1”

Виконав
студент групи ІК – 33:
Вересоцький А. Ю.

Перевірив:
асистент кафедри ІСТ
Нестерук А.О.

Київ 2025

**Завдання:**

**Завантаження набору даних**

За допомогою функції load_digits з модуля sklearn.datasets отримуємо об'єкт scikit-learn Bunch, що містить дані цифр і інформацію про набір даних Digits (так звані метадані):

```
from sklearn.datasets import load_digits
digits = load_digits()
```

Кількість зразків і ознак (на один зразок) підтверджується за допомогою атрибута shape масиву data, який показує, що набір даних складається з тисячі сімсот дев'яносто-сім рядків (зразків) і 64 стовпців (ознак):

```
digits.data.shape
```

```
(1797, 64)
```

Розміри масиву target підтверджують, що кількість цільових значень відповідає кількості зразків:

```
digits.target.shape
```

```
(1797,)
```

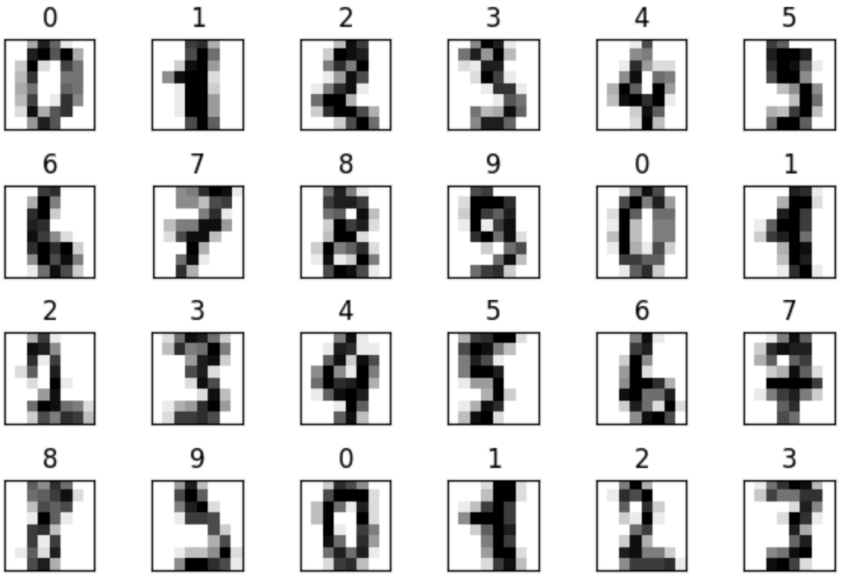
**1. Для дослідження даних, візуалізуйте їх. Виведіть зображення перших 24 і 36 цифр з набору**

Об'єкт Bunch, що повертається load_digits, містить атрибут images - масив, кожен елемент якого являє собою двовимірний масив 8 × 8 з інтенсивностями пікселів зображення цифри.

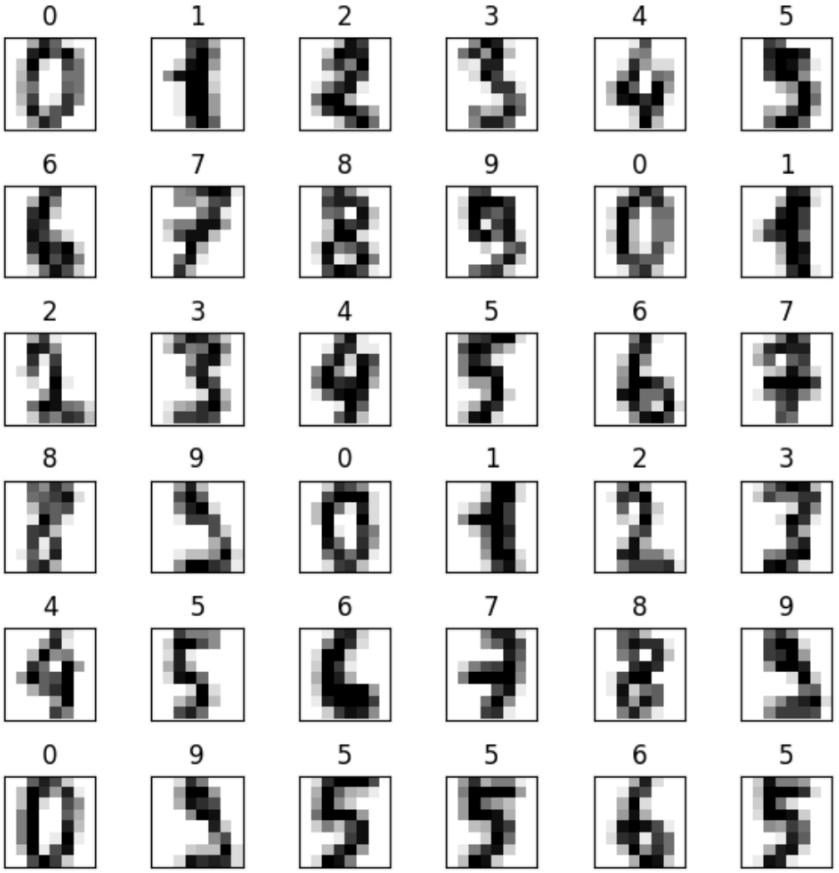
За допомогою функції matplotlib.pyplot.imshow візуалізуємо перші 24 та перші 36 цифр з набору:

```
figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(6, 4))
for item in zip(axes.ravel(), digits.images, digits.target):
    axes, image, target = item
    axes.imshow(image, cmap=plt.cm.gray_r)
    axes.set_xticks([])
    axes.set_yticks([])
    axes.set_title(target)
plt.tight_layout()
```

Перші 24 цифри:




Перші 36:



**2. Розбийте дані на навчальні та тестові, за замовчуванням train_test_split резервує 75% даних для навчання і 25% для тестування, змініть це.**

Додамо атрибут test_size=0.2 у виклик функції train_test_split, аби 20% даних були призначені для тестування, а інші 80% для тренування.

```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, random_state=11, test_size=0.2)
```

```
X_train.shape
```

```
(1437, 64)
```

```
X_test.shape
```

```
(360, 64)
```

**3. Створити та навчити модель**

Спочатку створюємо об'єкт оцінювача KneighborsClassifier та викликаємо метод fit об'єкта KNeighborsClassifier, який завантажує навчальний набір зразків (X_train) і навчальний набір цільових значень (y_train) в оцінювача:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

**4. Виконайте прогнозування класів**

```
predicted = knn.predict(X_test)
expected = y_test
```

**5. Порівняйте прогнозовані цифри з очікуваними для перших 36 тестових зразків:**

```
predicted[:36]
```

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 5, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5])
```

```
expected[:36]
```

```
array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 3, 6, 9, 6,
       0, 6, 9, 3, 2, 1, 8, 1, 7, 0, 4, 4, 1, 5])
```

## 6. Поясніть результат, застосуйте метрики точності моделі.

### 6.1 Метод score оцінювача

Кожен оцінювач містить метод score, який повертає оцінку результатів, показаних з тестовими даними, переданими в аргументах.

```
print(f'{knn.score(X_test, y_test):.2%}')
```

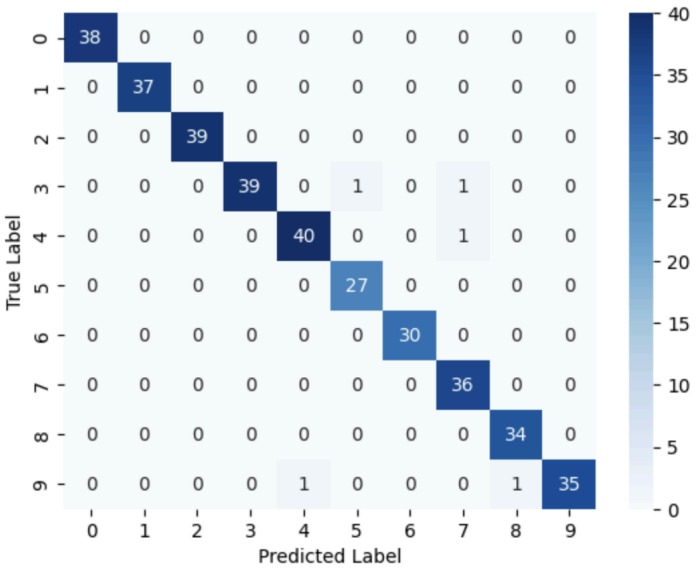
```
98.61%
```

### 6.2 Матриця невідповідностей.

Матриця невідповідностей містить інформацію про правильно і неправильно прогнозованих значеннях (також званих влученнями і промахами) для заданого класу.

```
confusion = confusion_matrix(expected, predicted)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", xticklabels=digits.target_names, yticklabels=digits.target_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
Text(50.722222222222214, 0.5, 'True Label')
```




## 7. Виведіть звіт класифікації

Модуль sklearn.metrics також надає функцію classification_report, яка виводить таблицю метрик классіфікаціі, заснованих на очікуваних і прогнозованих значеннях:

```
names = [str(digit) for digit in digits.target_names]
print(classification_report(expected, predicted, target_names=names))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 38 |
| 1 | 1.00 | 1.00 | 1.00 | 37 |
| 2 | 1.00 | 1.00 | 1.00 | 39 |
| 3 | 1.00 | 0.95 | 0.97 | 41 |
| 4 | 0.98 | 0.98 | 0.98 | 41 |
| 5 | 0.96 | 1.00 | 0.98 | 27 |
| 6 | 1.00 | 1.00 | 1.00 | 30 |
| 7 | 0.95 | 1.00 | 0.97 | 36 |
| 8 | 0.97 | 1.00 | 0.99 | 34 |
| 9 | 1.00 | 0.95 | 0.97 | 37 |
| accuracy | | | 0.99 | 360 |
| macro avg | 0.99 | 0.99 | 0.99 | 360 |
| weighted avg | 0.99 | 0.99 | 0.99 | 360 |

### 8. Використайте декілька моделей KNeighborsClassifier, SVC и GaussianNB для пошуку найкращої

Проводимо масштабування даних та навчання моделей:

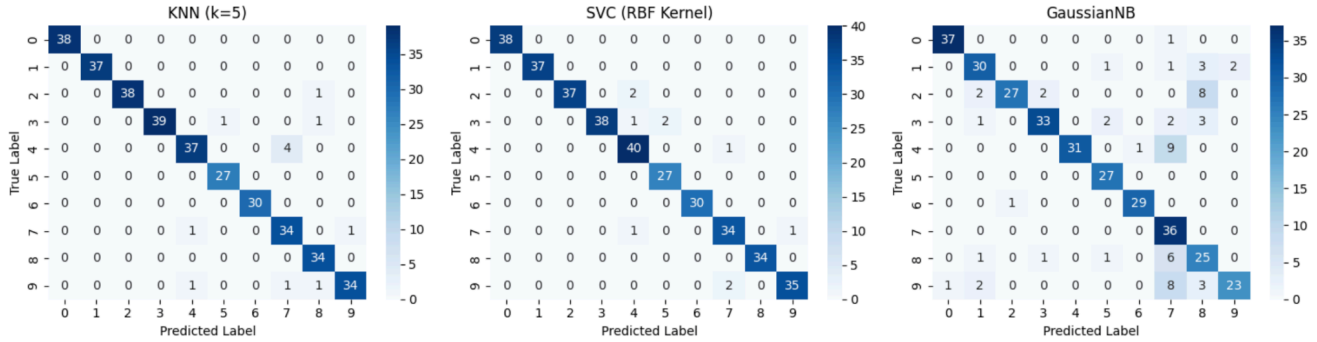
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

models = {
    "KNN (k=5)": KNeighborsClassifier(n_neighbors=5),
    "SVC (RBF Kernel)": SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42),
    "GaussianNB": GaussianNB()
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=digits.target_names, output_dict=True)

    results[name] = {
        "accuracy": acc,
        "precision": report["macro avg"]["precision"],
        "recall": report["macro avg"]["recall"],
        "f1-score": report["macro avg"]["f1-score"],
        "confusion_matrix": confusion_matrix(y_test, y_pred)
    }
```

Виводимо результати та візуалізуємо матрицю помилок:

```
for name, metrics in results.items():
    print(f"\n==== {name} ====")
    print(f"Accuracy: {metrics['accuracy']:.4f}")
    print(f"Precision: {metrics['precision']:.4f}")
    print(f"Recall: {metrics['recall']:.4f}")
    print(f"F1-score: {metrics['f1-score']:.4f}")

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

for ax, (name, metrics) in zip(axes, results.items()):
    sns.heatmap(metrics["confusion_matrix"], annot=True, fmt="d", cmap="Blues", ax=ax, xticklabels=digits.target_names, yticklabels=digits.tar
    ax.set_title(name)
    ax.set_xlabel("Predicted Label")
    ax.set_ylabel("True Label")

plt.tight_layout()
plt.show()
```

Результати:

```
==== KNN (k=5) ====
Accuracy: 0.9667
Precision: 0.9675
Recall: 0.9691
F1-score: 0.9678

==== SVC (RBF Kernel) ====
Accuracy: 0.9722
Precision: 0.9731
Recall: 0.9742
F1-score: 0.9732

==== GaussianNB ====
Accuracy: 0.8278
Precision: 0.8612
Recall: 0.8361
F1-score: 0.8345
```




- **SVC (RBF)** показав найкращі результати за всіма метриками — це очікувано, оскільки SVM добре працює на високовимірних даних.
- **KNN** працює дуже добре, хоча трохи гірше, ніж SVC. Це нормально, бо він чутливий до масштабування та вибору k.
- **GaussianNB** значно відстає — скоріш за все, через сильне припущення нормальності ознак, яке не дотримується у датасеті digits.

### 9. Налаштуйте гіперпараметр K в KNeighborsClassifier

Налаштуємо гіперпараметр K для KNeighborsClassifier з використанням перебору з крос-валідацією (GridSearchCV):

```python
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

param_grid = {
    'knn__n_neighbors': list(range(1, 16))
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Найкраще K:", grid_search.best_params_['knn__n_neighbors'])
print("Точність на валідації:", grid_search.best_score_)
print("Точність на тесті:", grid_search.score(X_test, y_test))
```

```
Найкраще K: 3
Точність на валідації: 0.9798296554394115
Точність на тесті: 0.9666666666666667
```

## Висновки

У ході виконання лабораторної роботи було розглянуто задачу багатокласової класифікації на основі вбудованого в бібліотеку Scikit-learn набору даних Digits. Було здійснено попередню обробку даних, їх візуалізацію, поділ на навчальну та тестову вибірки, а також реалізовано навчання та оцінювання кількох моделей машинного навчання: KNeighborsClassifier, SVC (з RBF ядром) та GaussianNB.

Була також проведена налаштування гіперпараметра k для моделі KNN з використанням крос-валідації, що дозволило покращити її ефективність.