#UpSkillWithKalpesh

Day 18

Data Science Unlocked

From Zero to Data Hero

Classification in ML
Part-1

Kalpesh Pathade
@DataSimplified

Classification in ML - Part 1

Type	@DataSimplified

1. What is Classification?

Classification is a **supervised learning** technique where a model learns to **categorize input data into predefined classes**. It is widely used in:

- **Spam Detection:** Classifying emails as spam or not.
- **Handwritten Digit Recognition:** Classifying images of handwritten numbers.
- **Medical Diagnosis:** Predicting diseases based on patient data.
- **Sentiment Analysis:** Categorizing text into positive, negative, or neutral.

Types of Classification:

- **Binary Classification**: Two possible classes (e.g., "Spam" vs. "Not Spam").
- **Multiclass Classification**: More than two classes (e.g., Digits 0-9).
- **Multilabel Classification**: Multiple labels can apply to one instance (e.g., Image tagging where one image has multiple objects).

---

2. Understanding the MNIST Dataset

The **MNIST dataset** consists of **70,000 grayscale images** of handwritten digits (0-9). Each image is **28×28 pixels** and flattened into a **784-dimensional feature vector**.

```
from sklearn.datasets import fetch_openml

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', as_frame=False)
```

```
# Extract features and labels
X, y = mnist.data, mnist.target

# Check the shape of data
print(X.shape)  # (70000, 784) → 70,000 images, each with 784 pixel values
print(y.shape)  # (70000,) → 70,000 labels
```

Why is X.shape (70000, 784)?

- Each **image is 28×28 pixels**, but ML models need 1D input.
- Flattening the image gives **784 features per sample**.

---

3. Visualizing an Image

We **reshape** the 1D array back to **28×28** and visualize it.

```
import matplotlib.pyplot as plt

# Function to visualize an image
def plot_digit(image_data):
    image = image_data.reshape(28, 28)  # Reshape from 1D (784,) to 2D (28×28)
    plt.imshow(image, cmap='binary')  # Binary colormap for black & white image
    plt.axis('off')  # Hide axis
    plt.show()

# Select and plot the first image
some_digit = X[0]
plot_digit(some_digit)

# Print the corresponding label
print("Label:", y[0])  # Should be the digit shown in the image
```

---

4. Splitting the Dataset

We use a **predefined split** in MNIST (first 60,000 for training, last 10,000 for testing).

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Why not shuffle?

- **MNIST is already shuffled**.
- If a dataset isn't shuffled, **shuffling is necessary** to ensure randomness in training.

How to shuffle if needed?

```
from sklearn.model_selection import train_test_split

# Randomly split 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42, shuffle=True)
```

When not to shuffle?

- If **time-series data** (e.g., stock prices, weather data).

---

5. Creating a Binary Classification Task

Let's **train a classifier to detect the digit "5"** (i.e., is the digit a "5" or not?).

```
# Convert labels to binary: True for '5', False for others
y_train_5 = (y_train == '5')
y_test_5 = (y_test == '5')
```

---

6. Choosing a Classifier (SGD Classifier)

Why SGDClassifier?

- **Works well for large datasets.**
- **Online learning**: Can update itself as new data arrives.
- **Fast for high-dimensional data** (like our 784-pixel features).

```
from sklearn.linear_model import SGDClassifier

# Train a Stochastic Gradient Descent (SGD) classifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

Making Predictions

```
# Predict if the first image is a "5"
print(sgd_clf.predict([some_digit]))  # Output: True/False
```

---

7. Evaluating Model Performance

Cross-Validation Accuracy

We use **cross-validation** to get a reliable accuracy score.

```
from sklearn.model_selection import cross_val_score

# Perform 3-fold cross-validation
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring='accuracy')
```

Potential issue with accuracy:

- If only **10% of images are "5"**, a model can always predict "Not 5" and still get **90% accuracy**.

---

8. Why Accuracy Is a Bad Metric?

Let's check a **DummyClassifier** that predicts "Not 5" all the time.

```
from sklearn.dummy import DummyClassifier

# Create a classifier that always predicts the most frequent class
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train_5)

# Cross-validation score
print(cross_val_score(dummy_clf, X_train, y_train_5, cv=3, scoring='accurac
y'))
```

Why is this bad?

- Even if the model gets **90% accuracy**, it **doesn't actually detect "5"s**.
- We need **better performance metrics** like:
  - **Precision & Recall**
  - **F1 Score**
  - **ROC-AUC Score**

---

9. What's Next?

In the next note, we will explore more evaluation for classification models.