Supervised Learning: Classification

Dr Ioanna Stamatopoulou

All material in these lecture notes is based on our textbook:
Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 3rd ed., O'Reilly, 2022

Outline

- Classification Types
  - Binary Classification
  - Multiclass Classification
  - Multilabel Classification
  - Multioutput Classification *(outside the scope of this module)*
- Performance Measures
  - Accuracy
  - Confusion Matrices
  - Precision and Recall
  - F_1 Score

Example Case Study

- Handwritten Digit classification
- The MNIST dataset - The “Hello World” of ML
- 70,000 images
- Each image has 28 x 28 pixels = 784 features
- Each feature is a value between 0 (white) and 255 (black)

- Visit our textbook’s Jupyter notebooks collection here: homl.info/colab3 and select *03_classification*

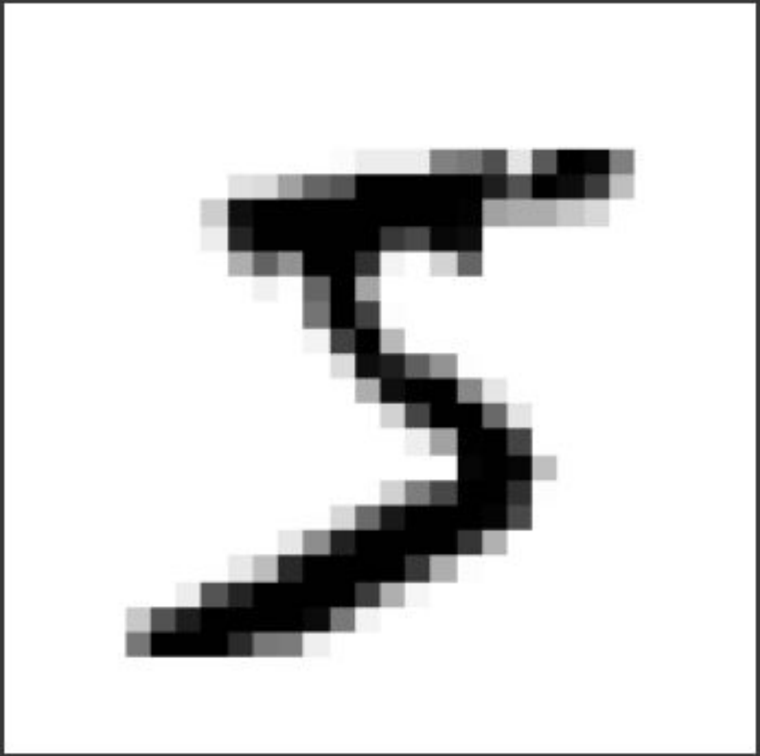
The MNIST dataset

Plotting one digit

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

some_digit = X[0]
plot_digit(some_digit)
save_fig("some_digit_plot")  # extra code
plt.show()
```



The MNIST dataset

Plotting an indicative number of digits

```python
# extra code – this cell generates and saves Figure 3–2
plt.figure(figsize=(9, 9))
for idx, image_data in enumerate(X[:100]):
    plt.subplot(10, 10, idx + 1)
    plot_digit(image_data)
plt.subplots_adjust(wspace=0, hspace=0)
save_fig("more_digits_plot", tight_layout=False)
plt.show()
```

Binary Classification

- Binary classification is about identifying instances that belong to **<u>one</u> target class ⇒ positive class**
- In other words, the aim is to **distinguish between two classes**
  - An instance either belongs to the target class or it doesn’t **⇒ negative class**
- In relation to MNIST: identify one digit
  - i.e. 5 and non-5

Performance Measure: Accuracy

- **Accuracy** measures the **ratio** (percentage) **of correct predictions**
- Evaluating the performance of such a classifier can be more challenging than evaluating a regressor!
  - If 10% of images are 5s and an algorithm classifies everything as a non-5, it will be correct 90% of the times!
- Accuracy is **not** the best performance measure for classifiers
  - Especially when dealing with **skewed datasets**, i.e. when some classes are more frequent than others, i.e. when your instances are far from equally distributed among the classes

Performance Measure: Confusion Matrix

- A Confusion Matrix counts how many times instances of a Class A are classified as Class B for all A-B pairs of classes
    - Rows represent the actual classes
    - Columns represent the predicted classes

```
array([[53892,   687],
       [ 1891,  3530]])
```

- Assuming a perfect performance, only the main diagonal has non-zero values

```
array([[54579,     0],
       [    0,  5421]])
```

Confusion Matrix

For the digit 5 of the MNIST dataset

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train_5, y_train_pred)
cm
```

```
array([[53892,   687],
       [ 1891,  3530]])
```

```
y_train_perfect_predictions = y_train_5  # pretend we reached perfection
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

```
array([[54579,     0],
       [    0,  5421]])
```

Performance Measure: Confusion Matrix

True Negatives: non-5s that were **correctly** classified as non-5s

		Predicted	
		non-5s	5s
Actual	non-5s	53892	687
	5s	1891	3530

False Positives: non-5s that were **incorrectly** classified as 5s

True Positives: 5s that were **correctly** classified as 5s

False Negatives: 5s that were **incorrectly** classified as non-5s

Performance Measures:
Confusion Matrix, Recall, Precision

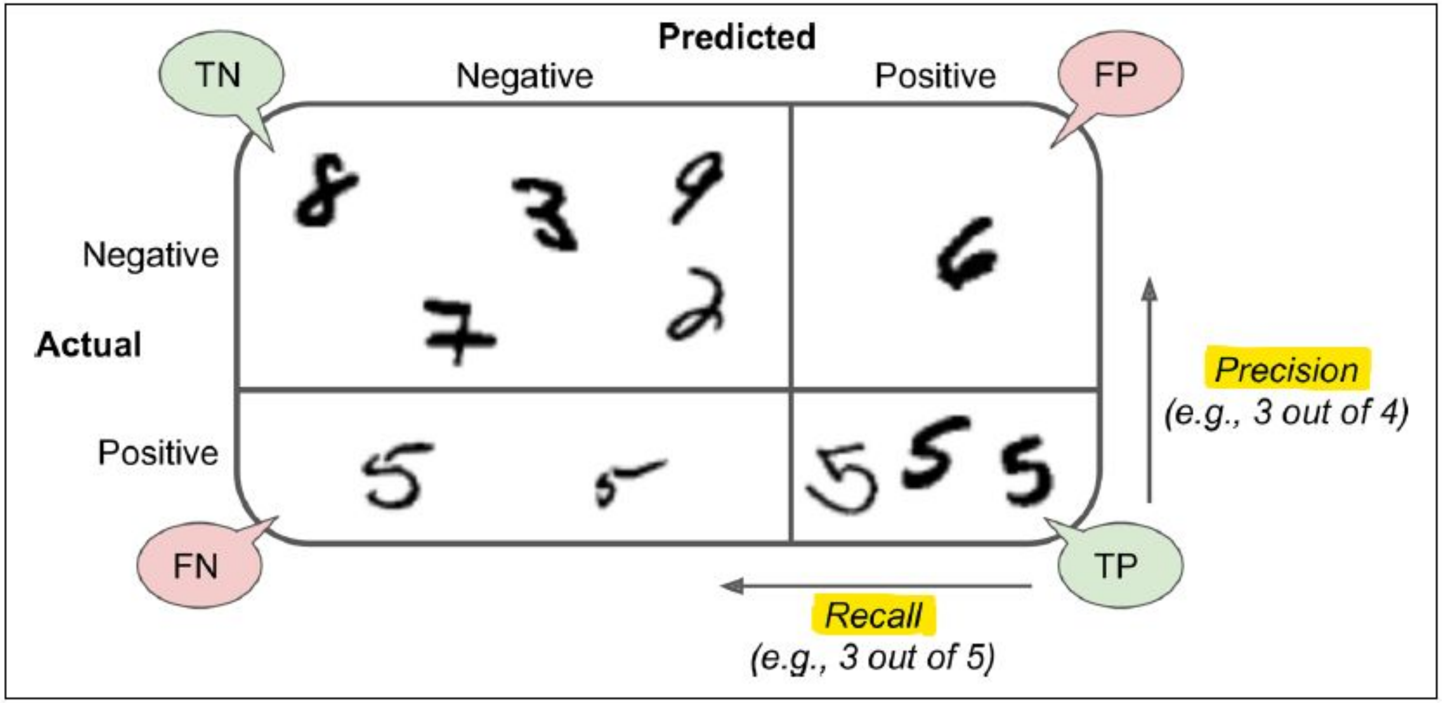


Figure 3-2. An illustrated confusion matrix shows examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

Terminology

Accuracy
The ratio of correct predictions

Confusion Matrix
It counts how many times instances of a Class A are classified as Class B for all A-B pairs of classes
i.e. all the true positive/negatives,
and all the false positives/negatives

Precision
The accuracy of the positive predictions

$$\text{precision} = \frac{TP}{TP + FP}$$

Recall or **Sensitivity** or **TPR (True Positive Rate)**
The ratio of positive instances that are correctly detected

$$\text{recall} = \frac{TP}{TP + FN}$$

Terminology

F_1 score

The **harmonic mean** of **precision** and **recall**

- for a high F_1 score, both precision as well as recall have to be high
- It favours models that have **similar** precision and recall

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Performance Measures: The Precision/Recall trade-off

- **Increasing precision reduces recall and vice-versa**
  - If someone says, “Let’s reach 99% precision,” you should ask, “At what recall?”
- Classifiers typically calculate a score for each instance using a **decision function**
- **If the score is higher than a threshold, the instance is classified in the positive class** (otherwise to the negative)
- The trade-off depends on the value of the **decision threshold**

Performance Measures:
The Precision/Recall trade-off

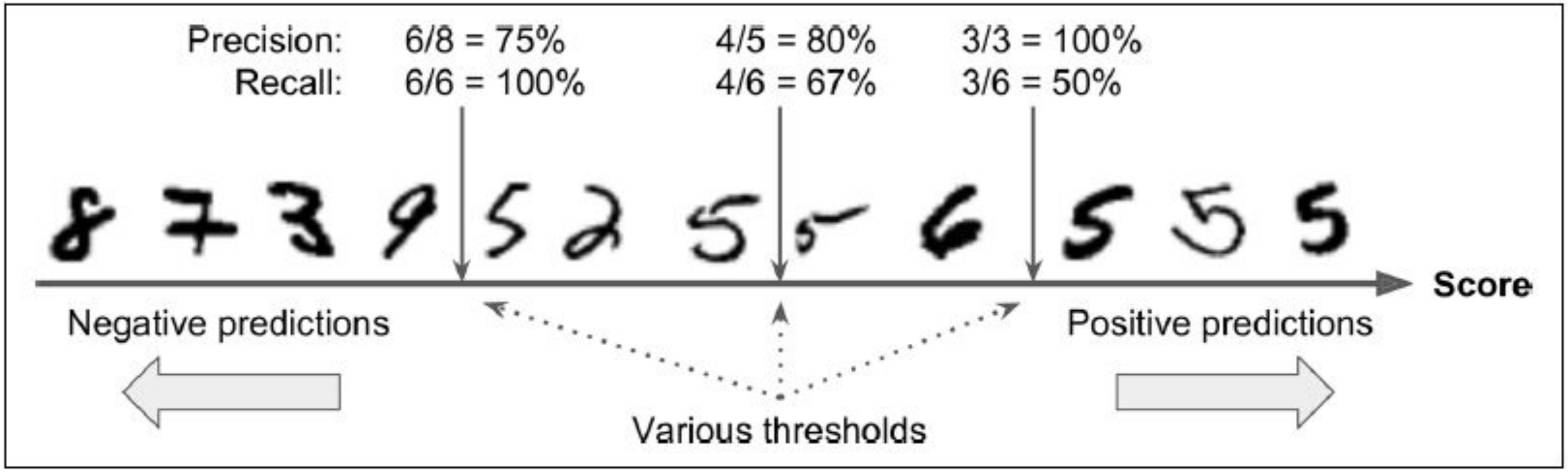
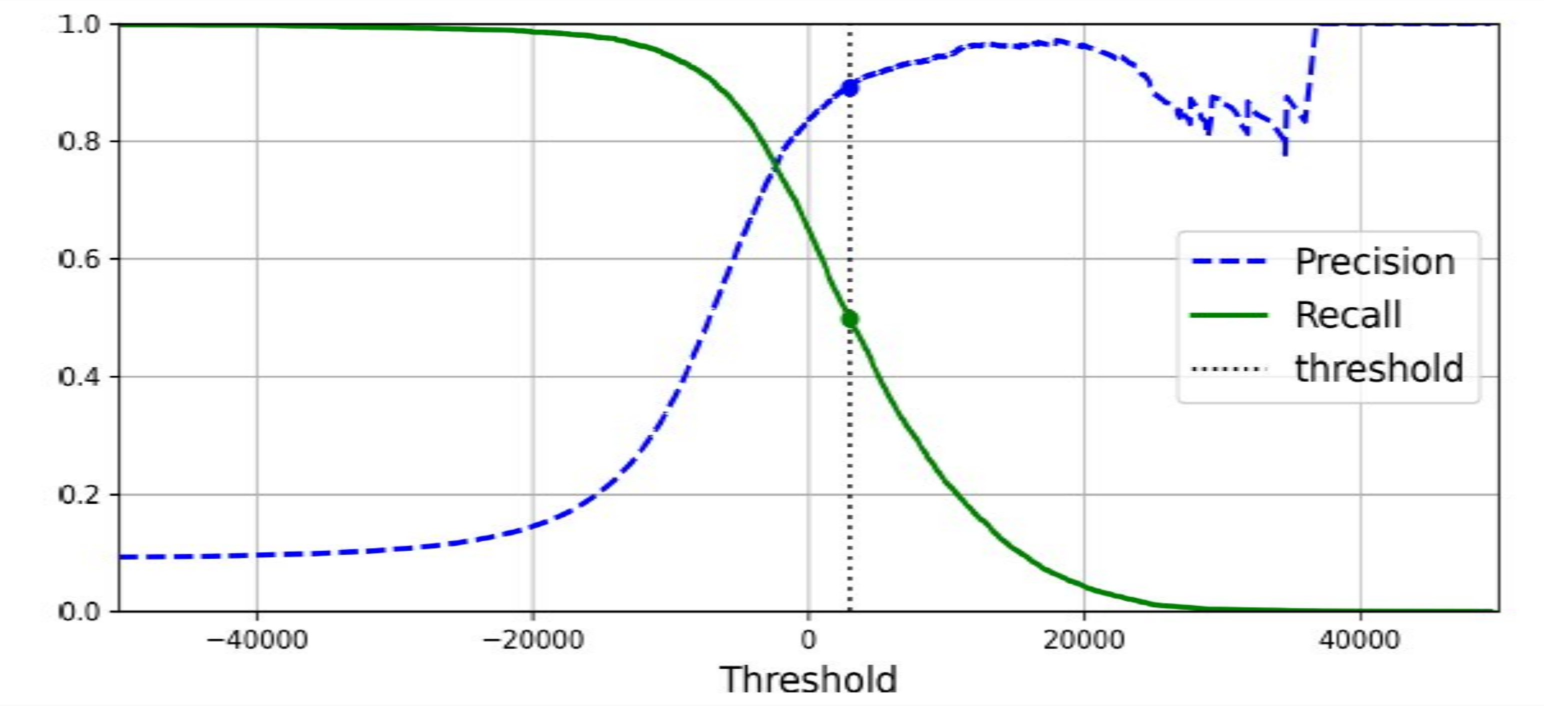


Figure 3-3. In this precision/recall trade-off, images are ranked by their classifier score, and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

Performance Measures: The Precision/Recall trade-off



Precision
Recall
F_1 Score

```
from sklearn.metrics import precision_score, recall_score

precision_score(y_train_5, y_train_pred)  # == 3530 / (687 + 3530)
```

```
0.8370879772350012
```

```
# extra code – this cell also computes the precision: TP / (FP + TP)
cm[1, 1] / (cm[0, 1] + cm[1, 1])
```

```
0.8370879772350012
```

```
recall_score(y_train_5, y_train_pred)  # == 3530 / (1891 + 3530)
```

```
0.6511713705958311
```

```
# extra code – this cell also computes the recall: TP / (FN + TP)
cm[1, 1] / (cm[1, 0] + cm[1, 1])
```

```
0.6511713705958311
```

```
from sklearn.metrics import f1_score

f1_score(y_train_5, y_train_pred)
```

```
0.7325171197343846
```

```
# extra code – this cell also computes the f1 score
cm[1, 1] / (cm[1, 1] + (cm[1, 0] + cm[0, 1]) / 2)
```

```
0.7325171197343847
```

Multiclass Classification

- **Multiclass** (or **multinomial**) classifiers distinguish between more than two classes
- Some classifiers are natively multiclass
  - Logistic Regression, RandomForest
- while others are strictly binary
  - SGDClassifier and SVC (C-Support Vector)
- BUT there are strategies for **using multiple binary classifiers to perform multiclass classification!**

Multiclass Classification using Binary Classifiers

OvA (or OvR) Strategy

- Train as many Binary Classifiers as your classes
  - For the Handwritten Digits problem you need **10**, one for each digit: a 0-detector, a 1-detector, …, a 9-detector
- To classify one instance:
  - Get the decision score from each classifier
  - Select the class with the highest score

    ⇒ **one-versus-all (OvA)**

    ⇒ or **one-versus-the-rest (OvR)**

Multiclass Classification using Binary Classifiers

OvO Strategy

For ***N*** classes you need
N x (N - 1) / 2
classifiers

- Train a Binary Classifier for every pair of classes
  - For the Handwritten Digits problem you need **45**,
    - one to distinguish between 0s and 1s
    - one to distinguish between 0s and 2s, **...,**
    - one to distinguish between 8s and 9s
- To classify one instance:
  - Select the class that wins more duels!

    ⇒ one-versus-one (OvO)
- **Advantage**: each classifier is trained only on part of the Training Set

Multiclass Classification using Binary Classifiers

OvA versus OvO

- OvA is generally preferred

- OvO is preferred in cases when an algorithm scales poorly with the size of the Training Set
  - e.g. Support Vector Machine classifiers

  $\Rightarrow$ Easier to train **many classifiers on small sets** rather than one/few classifiers on large sets

Multilabel Classification

- Multilabel classification is about identifying **multiple classes for each instance**
- The output of the classifier is an array of boolean tags
    - each position represents a particular class;
    - the value (true or false) whether the instance belongs to this class

Multilabel Classification

The MNIST dataset

Learning to identify whether:

- A number is greater or equal to 7
- A number is odd

some_digit is a 5:

- Less than 7
- Odd

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

K-nearest Neighbors supports multilabel classification

```
knn_clf.predict([some_digit])
```

```
array([[False,  True]])
```

Multilabel Classification Performance Measure

F_1 Score

- Compute F_1 score per class and average the scores
- Even better: Weigh each score in the average depending on how many instances belonging to each class exist in the set

Multilabel Classification

Chain Classification

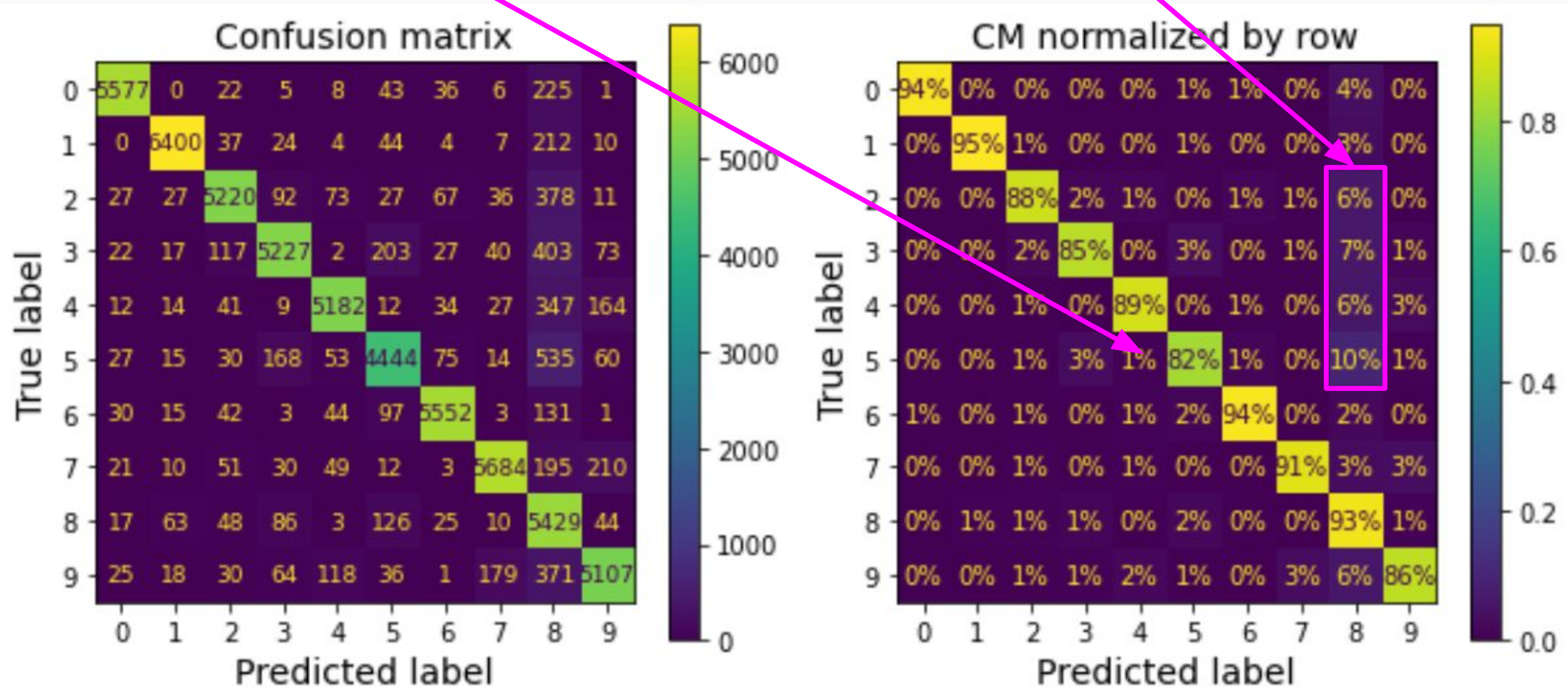
- You may wish to use a classifier that does <u>not</u> support multilabel classification:
- Organise the models in a **chain**:
- Each model in the chain uses
  - The input features of the instance, and
  - The predictions of all the models that come before it in the chain

Error Analysis using Confusion Matrices

Normalising the Confusion Matrix by Row (percentages instead of absolute instance numbers)

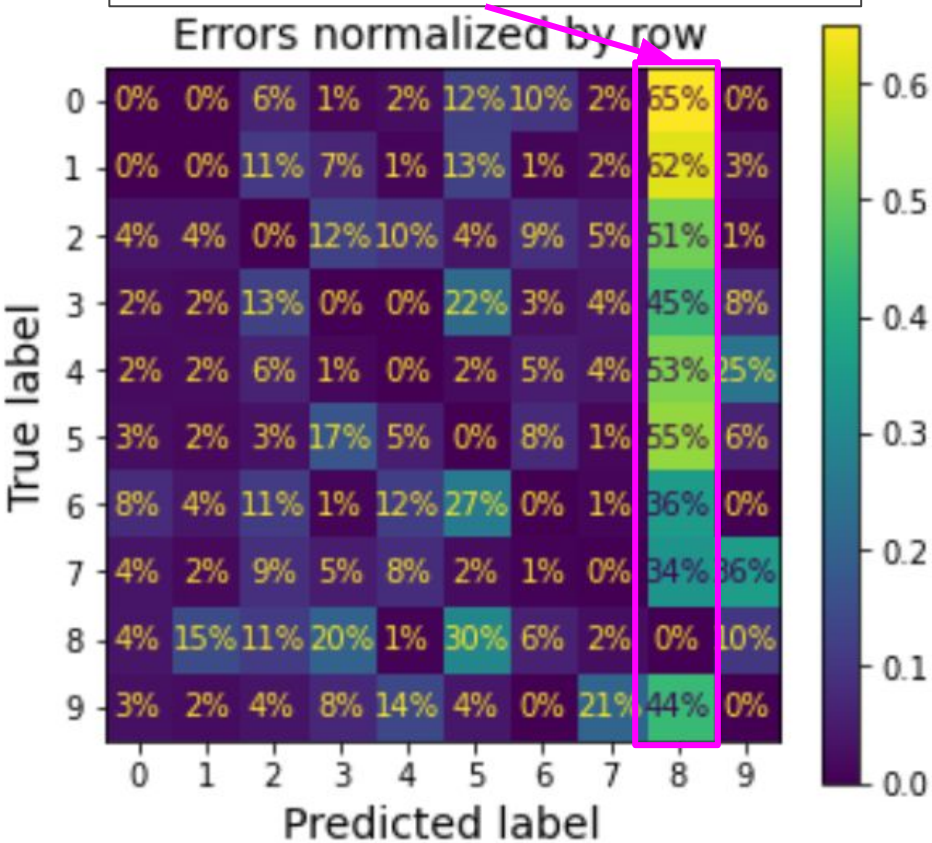
5s are the ones that are most often misclassified

A lot of numbers are misclassified as 8s

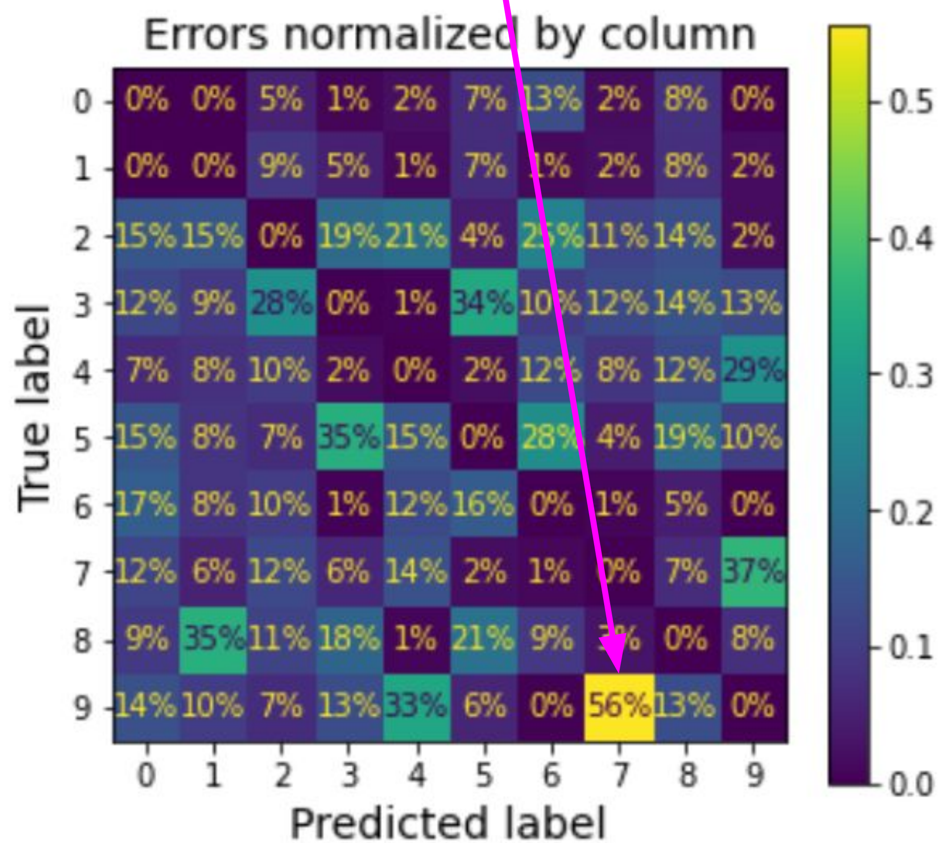


Normalising the **Error** by Row and Column (correct predictions are ignored)

- A lot of digits are misclassified as an 8
- 65% of **misclassified** 0s were classified as 8s, etc.



- 56% of wrongly predicted 7s were actually 9s



Using Error Analysis Results

- After you identify the types of errors your model performs, you can:
  - Gather more training data for the particular classes
  - Engineer new features that could help the classifier
  - Preprocess your data

⇒ **Data Augmentation**

Thank you!

Coming up next:
Supervised Learning: Classification Models