Confusion Matrix and Evaluation Metrics

Course:
INFO-6152 Deep Learning with Tensorflow & Keras 2

Developed by:
Mohammad Noorchenarboo

September 30, 2025

Current Section

Confusion Matrix for Binary Classification

Before extending to multi-class, let us review the binary case. Imagine a medical test classifying patients into “Positive” or “Negative”. How do we measure its performance?

Concept: Binary Confusion Matrix

The confusion matrix is 2×2:

$$CM = \begin{array}{l|c|c|} & \text{Predicted Negative} & \text{Predicted Positive} \\ \hline \text{Actual Negative} & TN & FP \\ \hline \text{Actual Positive} & FN & TP \\ \hline \end{array}$$

Metrics derived:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix for Binary Classification

Numerical Example

Suppose we test 10 patients:

$$CM = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

- Accuracy $= \frac{3+4}{10} = 0.7$
- Precision $= \frac{4}{4+1} = 0.8$
- Recall $= \frac{4}{4+2} \approx 0.667$
- F1-score $= \frac{2 \cdot 0.8 \cdot 0.667}{0.8+0.667} \approx 0.727$

Confusion Matrix for Binary Classification

Binary Confusion Matrix Example

```python
import numpy as np
from sklearn.metrics import confusion_matrix, precision_score,
     recall_score, f1_score

# True labels
y_true = [0,0,0,0,1,1,1,1,1,1]

# Predicted labels
y_pred = [0,0,1,0,1,1,0,1,1,0]

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)
# Expected [[3 1]
#           [2 4]]

print("Precision:", precision_score(y_true, y_pred))
print("Recall:", recall_score(y_true, y_pred))
print("F1-score:", f1_score(y_true, y_pred))
```

Get more info → scikit-learn Confusion Matrix

Summary: Confusion Matrix for Binary Classification

Aspect	**Explanation**
Confusion Matrix	2×2 table: TN, FP, FN, TP
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$, correctness of positive predictions
Recall	$\frac{TP}{TP+FN}$, ability to detect positives
F1-score	Harmonic mean of Precision and Recall
Python Code	Demonstrates binary confusion matrix and metrics with sklearn

Current Section

Confusion Matrix for Multi-Class Classification

In healthcare, imagine an AI system classifying images into **3 classes**: “Healthy”, “Pneumonia”, and “COVID-19”. How do we measure how well the model predicts each category? The confusion matrix is the tool for this.

Concept: Multi-Class Confusion Matrix

For n classes, the confusion matrix is an $n \times n$ table. Entry (i,j) shows how many samples from true class i were predicted as class j.

$$CM = \begin{bmatrix} cm_{11} & cm_{12} & \dots & cm_{1n} \\ cm_{21} & cm_{22} & \dots & cm_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ cm_{n1} & cm_{n2} & \dots & cm_{nn} \end{bmatrix}$$

Metrics derived:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} cm_{ii}}{\sum_{i=1}^{n} \sum_{j=1}^{n} cm_{ij}}$$

For a given class m:

$$\text{Precision}_m = \frac{cm_{mm}}{\sum_{i=1}^{n} cm_{im}}, \quad \text{Recall}_m = \frac{cm_{mm}}{\sum_{j=1}^{n} cm_{mj}}$$

Confusion Matrix for Multi-Class Classification

Build Confusion Matrix with TensorFlow/Keras Predictions

```python
import numpy as np
import tensorflow as tf
from sklearn.metrics import confusion_matrix, precision_score,
    recall_score

# Example true labels (ground truth)
y_true = [0,0,0,0,1,1,1,1,1,2,2,2]  # 0:Healthy, 1:Pneumonia,
    2:COVID-19

# Example predicted labels (normally from model.predict)
y_pred = [0,0,0,1,1,1,1,1,2,1,2,2]

# Build confusion matrix
cm = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:\n", cm)
# Expected:
# [[3 1 0]
#  [0 4 1]
#  [0 1 2]]
```

Confusion Matrix for Multi-Class Classification

Compute Accuracy, Precision and Recall Per Class

```python
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)

# Calculate per-class precision and recall
# average=None -> returns the metric separately for each class
# instead of aggregating (macro/micro/weighted)
precisions = precision_score(y_true, y_pred, average=None)
recalls = recall_score(y_true, y_pred, average=None)

# Print results
print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
print("Precision per class:", precisions)
print("Recall per class:", recalls)
# Expected output:
# Confusion Matrix:
# [[3 1 0]
#  [0 4 1]
#  [0 1 2]]
# Accuracy: 0.75
# Precision per class: [1.         0.66666667 0.66666667]
# Recall per class:    [0.75 0.8  0.67]

# In practice:
# y_pred = np.argmax(model.predict(x_test), axis=1)
# This is where TensorFlow/Keras provides predictions.
```

Confusion Matrix for Multi-Class Classification

Numerical Example

Suppose we classify 12 samples into 3 classes:

$$CM = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 4 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

- Accuracy $= \frac{cm_{11}+cm_{22}+cm_{33}}{\sum_{i=1}^{3}\sum_{j=1}^{3} cm_{ij}} = \frac{3+4+2}{3+1+0+0+4+1+0+1+2}$
- For class 1: Precision$_1 = \frac{cm_{11}}{cm_{11}+cm_{21}+cm_{31}} = \frac{3}{3+0+0}$,
  Recall$_1 = \frac{cm_{11}}{cm_{11}+cm_{12}+cm_{13}} = \frac{3}{3+1+0}$
- For class 2: Precision$_2 = \frac{cm_{22}}{cm_{12}+cm_{22}+cm_{32}} = \frac{4}{1+4+1}$,
  Recall$_2 = \frac{cm_{22}}{cm_{21}+cm_{22}+cm_{23}} = \frac{4}{0+4+1}$
- For class 3: Precision$_3 = \frac{cm_{33}}{cm_{13}+cm_{23}+cm_{33}} = \frac{2}{0+1+2}$,
  Recall$_3 = \frac{cm_{33}}{cm_{31}+cm_{32}+cm_{33}} = \frac{2}{0+1+2}$

Get more info $\rightarrow$ scikit-learn Confusion Matrix

Summary: Confusion Matrix for Multi-Class Classification

Aspect	**Explanation**
Definition	$n \times n$ matrix where cm_{ij} = true class i predicted as class j
Accuracy	$\frac{\text{sum of diagonal}}{\text{total samples}}$
Precision$_m$	$\frac{cm_{mm}}{\sum_i cm_{im}}$, correctness of predictions for class m
Recall$_m$	$\frac{cm_{mm}}{\sum_j cm_{mj}}$, ability to detect class m correctly
Numerical Example	For 3-class case, $Accuracy = 0.75$, class-wise Precision/Recall computed
Python Code	Demonstrates confusion matrix, precision, recall using Keras predictions + scikit-learn metrics

Current Section

Hierarchical Information in CNNs

Consider a CNN trained on chest X-ray images to detect diseases. How does the network go from raw pixels to a decision like “Pneumonia”? The answer: **hierarchical feature extraction**.

Concept: Hierarchical Representation

Each CNN layer extracts progressively more abstract features:

- Early layers detect **local patterns**: edges, corners, simple textures.
- Middle layers combine these into **shapes or motifs**: circles, curves, small anatomical structures.
- Deeper layers integrate patterns into **semantic features**: lungs, lesions, or abnormalities.

Formally, with input x and stacked convolutional layers $f^{(l)}$:

$$h^{(1)} = f^{(1)}(x), \quad h^{(2)} = f^{(2)}(h^{(1)}), \quad \ldots, \quad h^{(L)} = f^{(L)}(h^{(L-1)})$$

Each $h^{(l)}$ encodes features at level l of abstraction.

Interactive example $\rightarrow$ link

Hierarchical Information in CNNs

Numerical/Conceptual Example

In a CNN trained on digits:

- Layer 1 might respond strongly to horizontal/vertical edges.
- Layer 2 combines edges into strokes or loops.
- Layer 3 assembles loops and strokes into full digit shapes.

This hierarchy is why CNNs outperform simple feature extractors.

Interactive example → link

Caveats

- Deeper is not always better: too many layers without enough data can cause overfitting.
- Interpretability decreases as layers go deeper.

Get more info → CS231n Convolutional Networks Notes

Summary: Hierarchical Information in CNNs

Aspect	**Explanation**
Early Layers	Detect simple features like edges, corners, textures
Middle Layers	Combine features into patterns like shapes, motifs
Deeper Layers	Capture high-level semantic structures (objects, organs)
Formal Representation	$h^{(l)} = f^{(l)}(h^{(l-1)})$
Example	From strokes $\rightarrow$ loops $\rightarrow$ digits
Caveats	Overfitting risk, reduced interpretability with depth