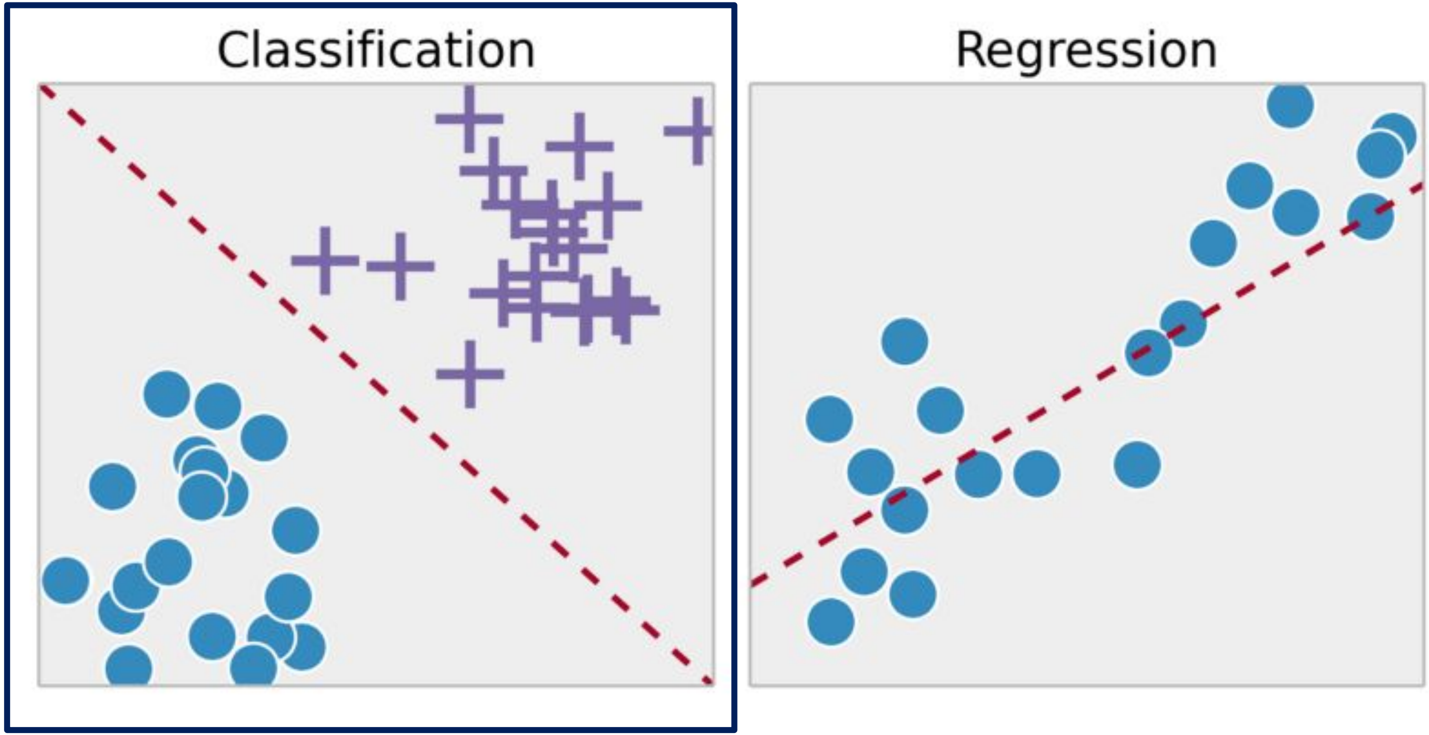
Classification
Logistic Regression
& Classification Metrics

November 26, 2023

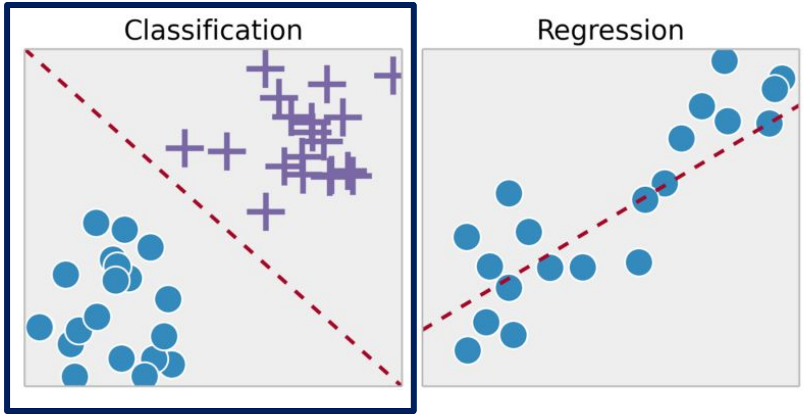
1. What is Classification?

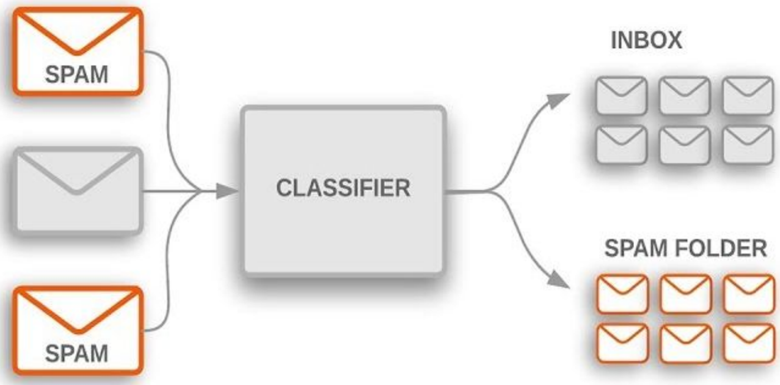

Classification
Regression

Classification vs Regression

- In **Regression**, we predict a continuous variable
- In **Classification**, we predict a discrete variable, usually a label

- **Example**: predicting whether an email is spam
  - **Labels**: Spam / Not spam
  - **Features**: Sender address, keywords in the text, time of day, regular contact, etc.





Binary vs Multi-label classification

Breast cancer prediction - Binary

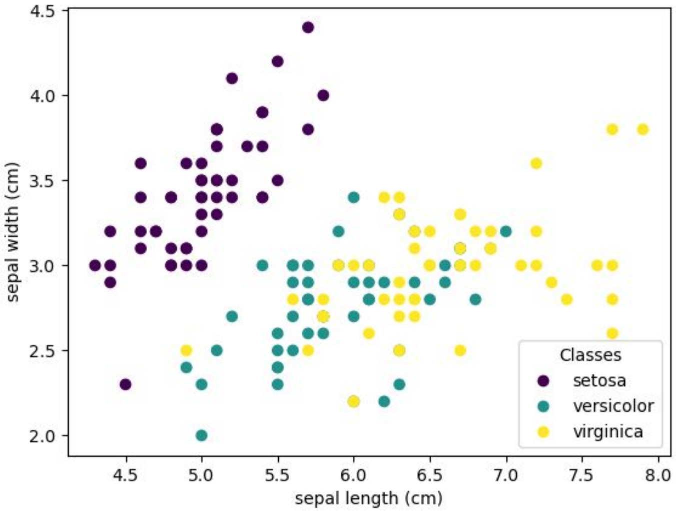
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...
...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target,
```

Binary vs Multiclass classification

Flower classification (Iris) - Multiclass

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8



2. Logistic Regression

How to build a model?

- Can we use a **linear regression**?

$$\hat{y} = b + w_1 x_1 + \cdots + w_n x_n$$

- The equation goes from **-∞** to **+∞**
- We want something that goes from **0** to **1** (representing the probability of our class)

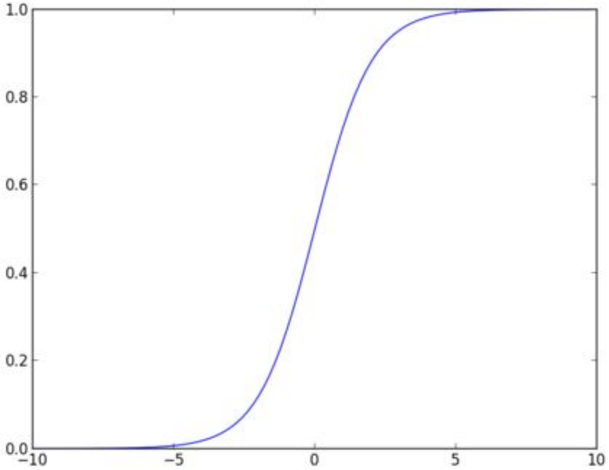
How to build a model?

- Can we use a **linear regression**?

$$\hat{y} = b + w_1 x_1 + \cdots + w_n x_n$$

- The equation goes from **-∞** to **+∞**
- We want something that goes from **0** to **1** (representing the probability of our class)
- Introducing the **Logistic function**

$$\hat{y} = \frac{1}{1+e^{-z}}$$



How to build a model?

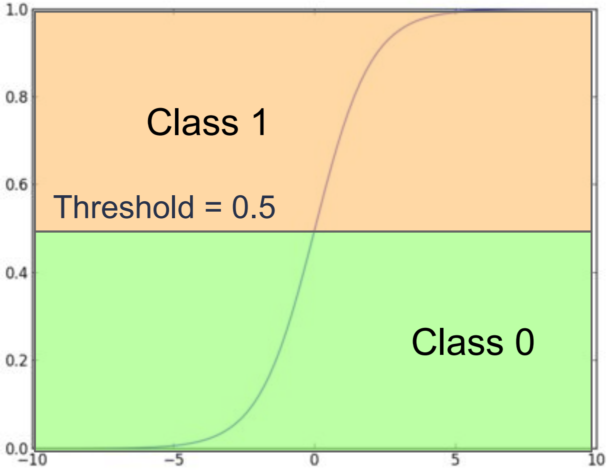
- Can we use a **linear regression**?

$$\hat{y} = b + w_1 x_1 + \cdots + w_n x_n$$

- The equation goes from **-∞** to **+∞**
- We want something that goes from **0** to **1** (representing the probability of our class)
- Introducing the **Logistic function**

$$\hat{y} = \frac{1}{1+e^{-z}}$$

$$z = b + w_1 x_1 + \cdots + w_n x_n$$

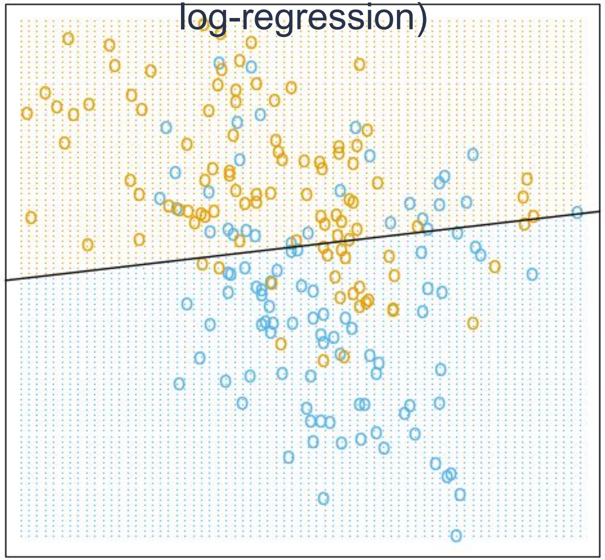


- This is the basis for **Logistic Regression**
- We can solve a **binary classification problem** by **optimizing a linear regression**

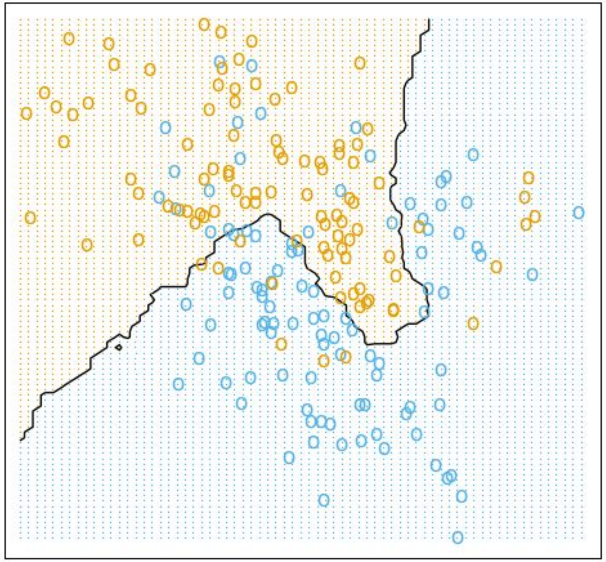
Decision boundary

- How does a classifier look if we look at just two features?

Linear classifier (e.g. log-regression)



Non-linear classifier (e.g. KNN)

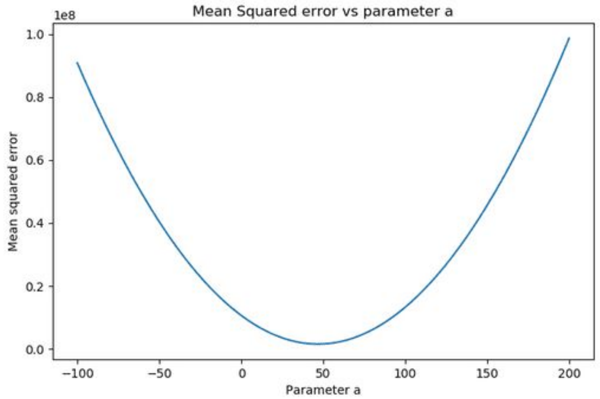


3. Optimization

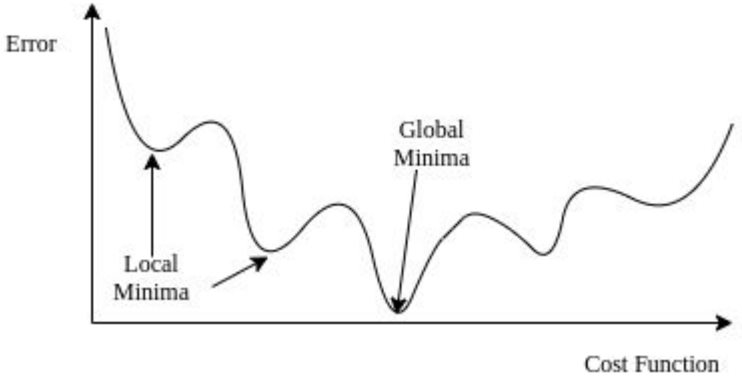
How do we optimize the parameters

- For **linear regression** we simply used the **mean-squared error (MSE)** as the cost

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y)^2$$



- However, this function is not **always convex** for the logistic function
    - → This can lead us to pick the incorrect parameters that minimize our error



Logistic loss

- We can use the **logistic loss**, also known as the **negative log-likelihood**

$$L(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{, if } y = 1 \\ -\log(1-\hat{y}) & \text{, if } y = 0 \end{cases}$$

Logistic loss

- We can use the **logistic loss**, also known as the **negative log-likelihood**

$$L(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{, if } y = 1 \\ -\log(1-\hat{y}) & \text{, if } y = 0 \end{cases}$$

- The **cost function** is the the sum of the loss over all samples

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$$

Logistic loss

- We can use the **logistic loss**, also known as the **negative log-likelihood**

$$L(\hat{y}, y) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{, if } y = 1 \\ -\log(1-\hat{y}) & \text{, if } y = 0 \end{cases}$$

- The **cost function** is the the sum of the loss over all samples

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$$

- We use an **optimization algorithm** to calculate the parameters, e.g. **gradient descent**

$$w_i \rightarrow w_i - l_r \frac{\partial J}{\partial w_i}$$

Logistic loss

- We can use the **logistic loss**, also known as the **negative log-likelihood**

$$L(\hat{y}, y) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$$

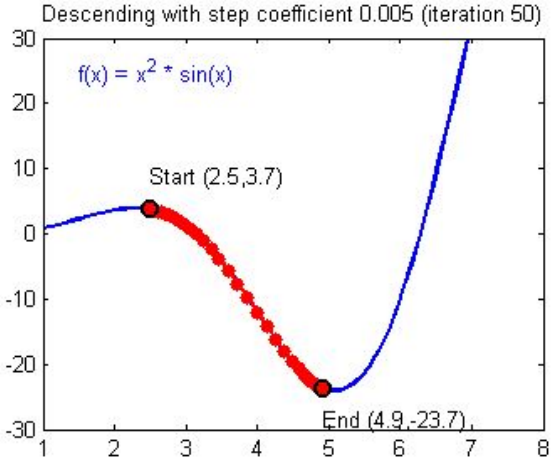
$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{, if } y = 1 \\ -\log(1-\hat{y}) & \text{, if } y = 0 \end{cases}$$

- The **cost function** is the the sum of the loss over all samples

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$$

- We use an **optimization algorithm** to calculate the parameters, e.g. **gradient descent**

$$w_i \rightarrow w_i - l_r \frac{\partial J}{\partial w_i}$$

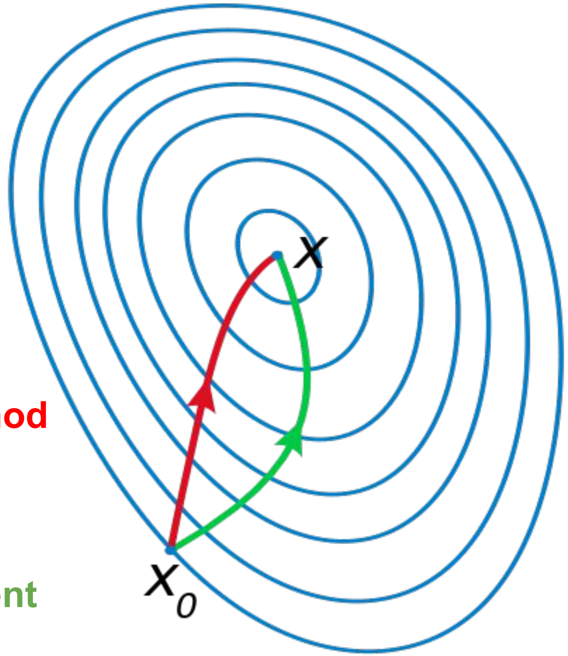


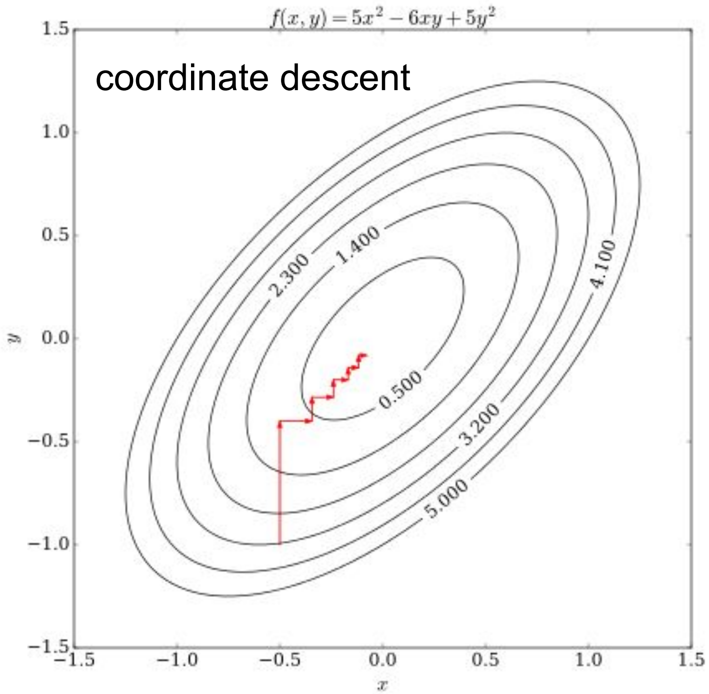
Optimization algorithms - measure the steepness of the landscape (1st and 2nd derivative)

Newton's method

vs.

gradient descent





4. SKLearn example

Importing data

```python
import pandas as pd
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True, as_frame=True)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...
...	...	...	...	...	...	...	...	...	...	...	...

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target,
```

Splitting the data into train and test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Scaling the data

$$x_i^{scaled} = \frac{x_i - \bar{x}}{\sigma_x}$$

Scaling the data

$$x_i^{scaled} = \frac{x_i - \bar{x}}{\sigma_x}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Scaling the data

$$x_i^{scaled} = \frac{x_i - \bar{x}}{\sigma_x}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
(array([[ 0.35581987, -1.72664707,  0.41303118, ...,  1.05652305,
          0.51844283,  0.98925483],
        [-0.37326674,  0.50881308, -0.39469992, ..., -0.89667412,
         -0.63236707, -0.11469617],
        [ 1.4033581 , -0.15144941,  1.34799923, ...,  1.46353548,
          1.02587412, -0.03369823],
        ...,
        [ 0.02898794, -0.56411347, -0.0829084 , ..., -1.2463348 ,
         -0.67979055, -1.26509354],
        [-0.05760855,  0.09614902, -0.04957475, ...,  1.05652305,
          0.45995387,  1.24414347],
        [-0.5604269 ,  0.30837625, -0.61462066, ..., -0.6145405 ,
         -0.30672581, -0.8357478 ]]),
```

Training the logistic regression

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

Training the logistic regression

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

```
y_pred = logistic_model.predict(X_test_scaled)
```

```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1])
```

Training the logistic regression

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

```
y_pred = logistic_model.predict(X_test_scaled)
```

```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1])
```

```
y_proba = logistic_model.predict_proba(X_test_scaled)
```

```
array([[1.36008828e-01, 8.63991172e-01],
       [9.99977295e-01, 2.27049622e-05],
       [9.96080057e-01, 3.91994328e-03],
       [7.80003100e-04, 9.99219997e-01],
       [1.14294099e-04, 9.99885706e-01],
       [1.00000000e+00, 3.50087563e-10],
       [9.99999993e-01, 7.08653260e-09],
       [9.54663200e-01, 4.53368002e-02],
       [4.31753581e-01, 5.68246419e-01],
       [1.14096940e-03, 9.98859031e-01],
       [5.99262279e-02, 9.40073772e-01],
       [9.84076010e-01, 1.59239901e-02],
       [7.18535007e-03, 9.92814650e-01],
       [8.28290034e-01, 1.71709966e-01],
       [2.59614386e-03, 9.97403856e-01],
```

How good is our model?

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9787234042553191
```

- Pretty good!

How good is our model?

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9787234042553191
```

- Pretty good!

```
y_pred_train = logistic_model.predict(X_train_scaled)
accuracy_score(y_train, y_pred_train)
```

```
0.9868766404199475
```

- Why did our accuracy fall between training and testing?

Recap

- **Classification** attempts to predict, for each individual in a population, to **which class each individual belongs to.**

- A classification problem can be either be **binary** or **multiclass**.

- **Logistic Regression** is a linear classification algorithm that outputs class probabilities.

- Logistic Regression uses the **negative log-likelihood** cost function to optimize parameters

5. Classification metrics

The problem with accuracy

- Let’s import another dataset for a blood transfusion centre, whose target is **whether an individual donated blood or not** (binary),

```
from sklearn.datasets import fetch_openml

X, y = fetch_openml(data_id=1464, return_X_y=True, parser="pandas")
y = y.replace({"1": 0, "2": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_state=42)

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)

y_pred = logistic_model.predict(X_test_scaled)
```

The problem with accuracy

- How good is our model?

```
accuracy_score(y_test, y_pred)
```

```
0.7540106951871658
```

The problem with accuracy

- How good is our model?

```
accuracy_score(y_test, y_pred)
```

```
0.7540106951871658
```

- Not too bad… But what if we always predicted the label “0”?

The problem with accuracy

- How good is our model?

```
accuracy_score(y_test, y_pred)
```

```
0.7540106951871658
```

- Not too bad… But what if we always predicted the label “0”?

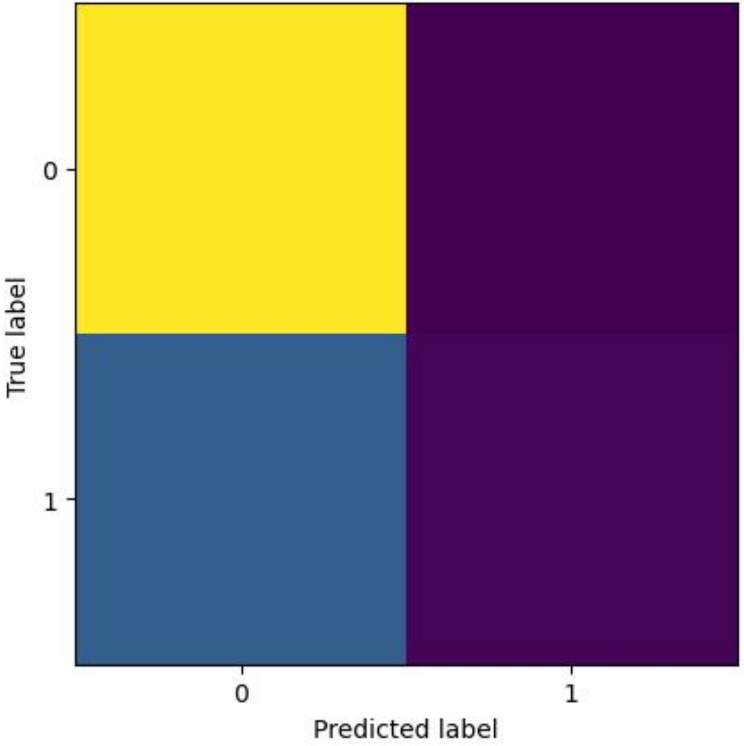
```
import numpy as np
y_all_zeros = np.zeros_like(y_test)
accuracy_score(y_test, y_all_zeros)
```

```
0.7433155080213903
```

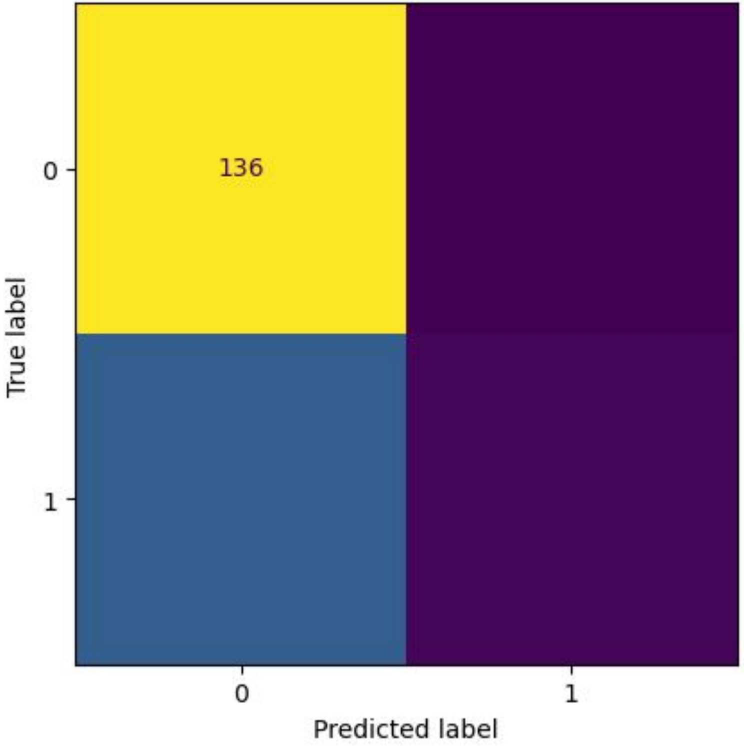
- That’s not much worse than our model… What’s going on?
- Can we have a **more detailed look** at our predictions for **both classes**?

6. Confusion matrix

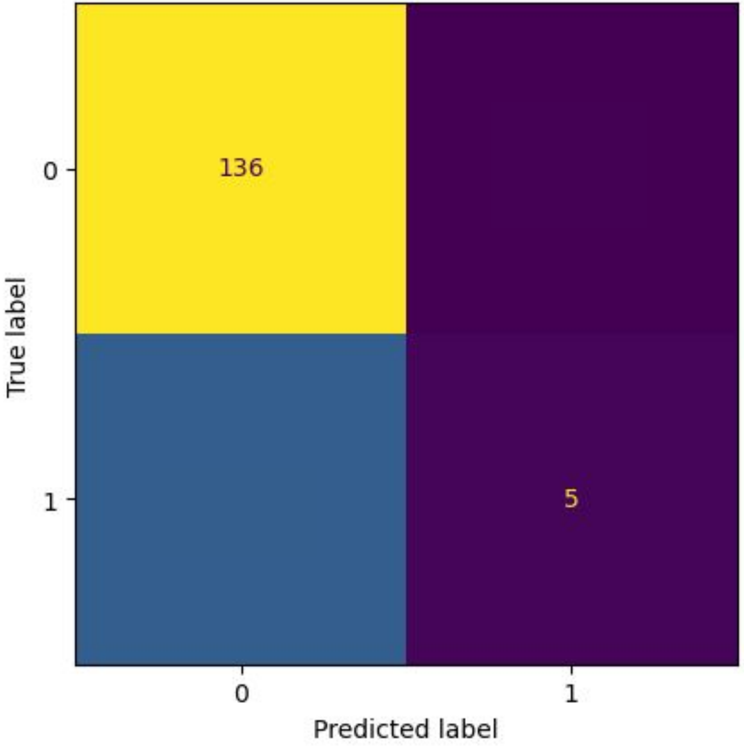
The confusion matrix



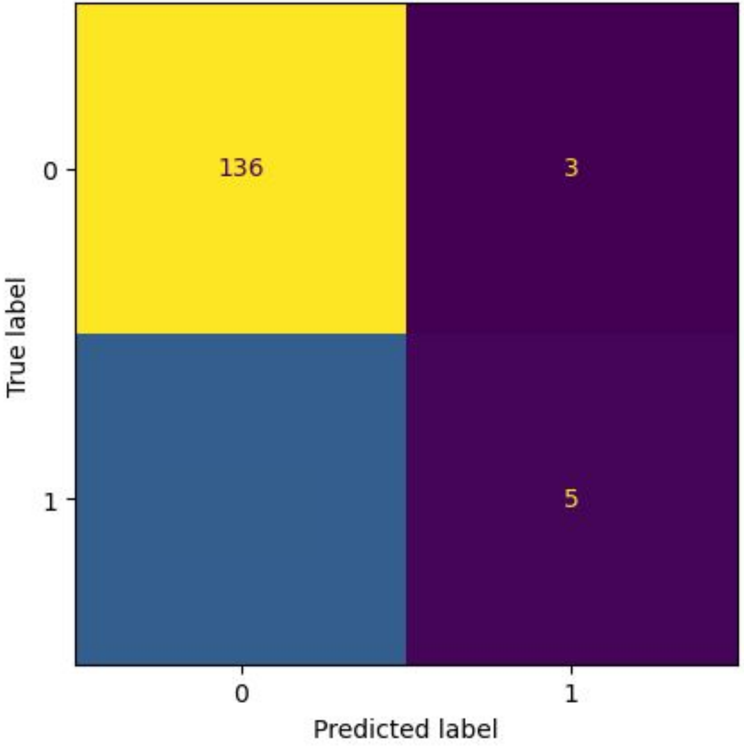
The confusion matrix



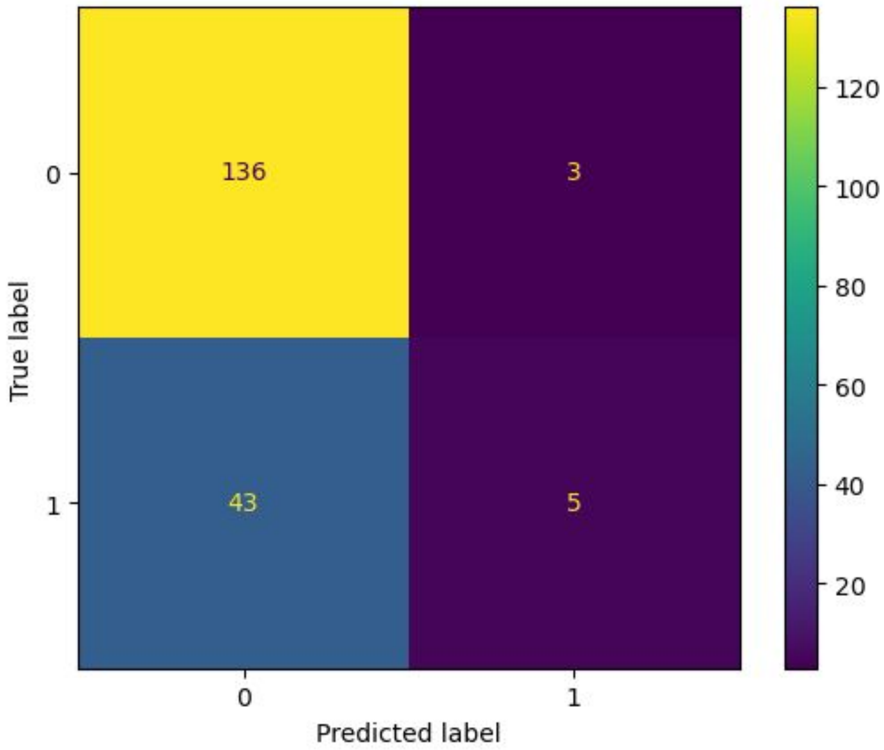
The confusion matrix



The confusion matrix

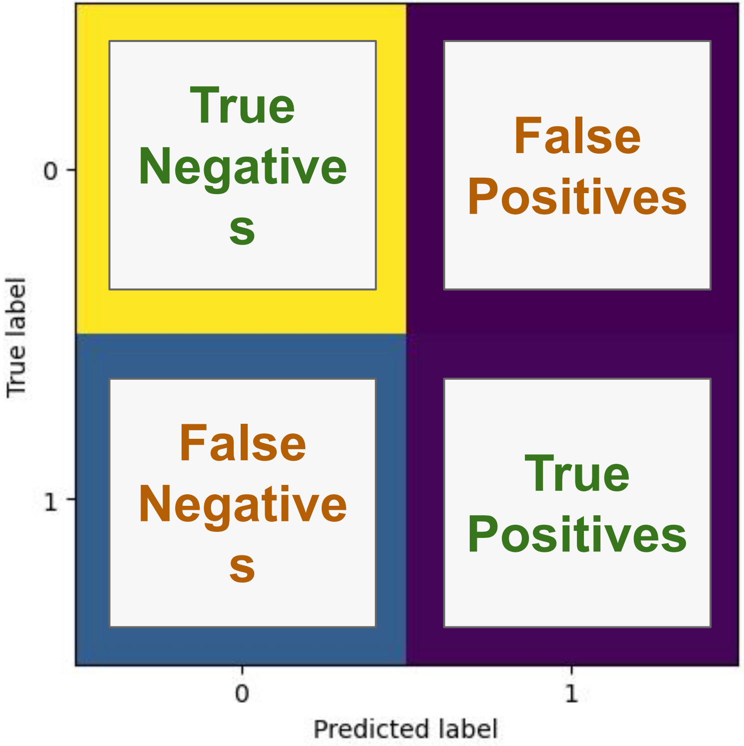


The confusion matrix



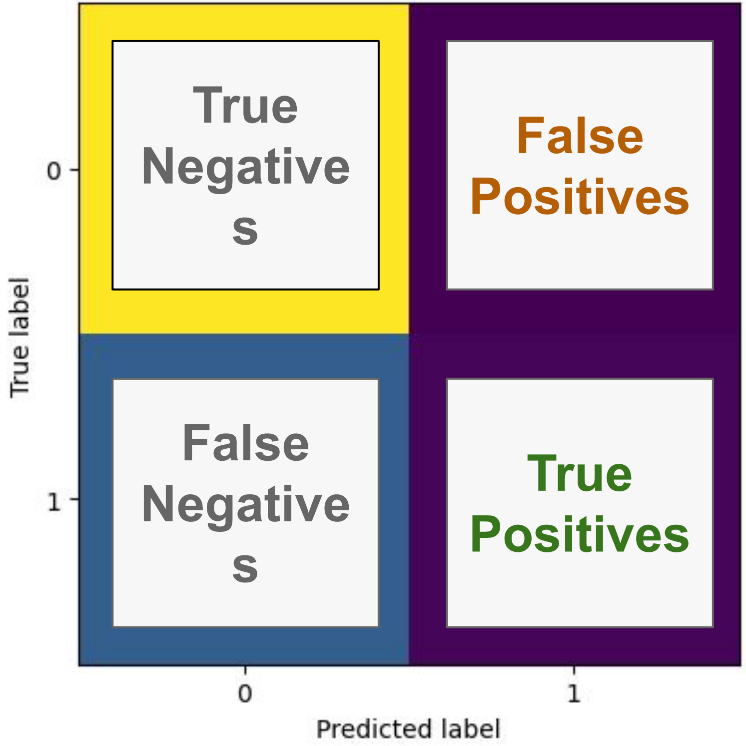

```
from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
)
cm = confusion_matrix(y_test, y_pred)
cm_display = ConfusionMatrixDisplay(cm).plot()
```

The confusion matrix



7. Precision, Recall & F1-score

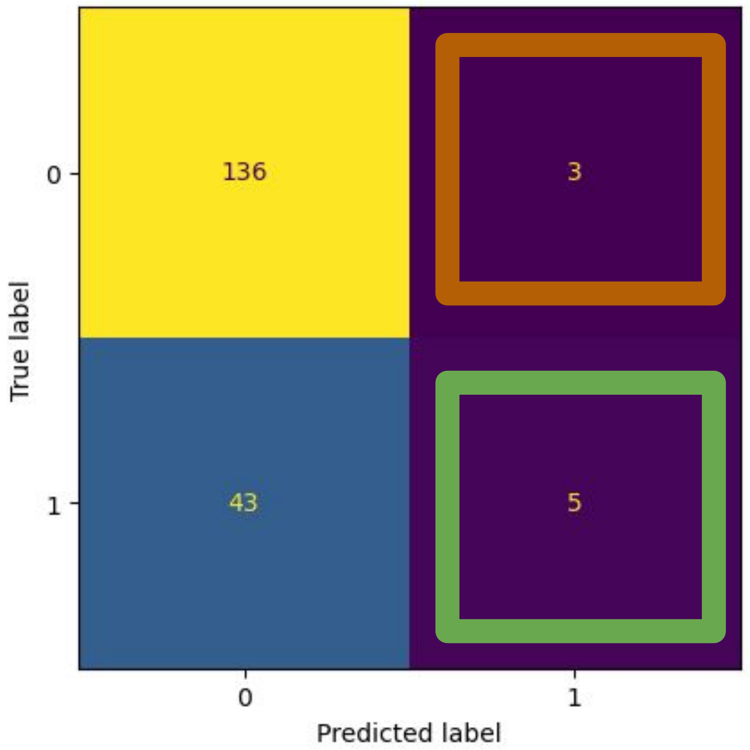
Precision



$$P = \frac{\boxed{TP}}{\boxed{TP} + \boxed{FP}}$$

*What proportion of positive **identifications** was actually correct?*

The confusion matrix



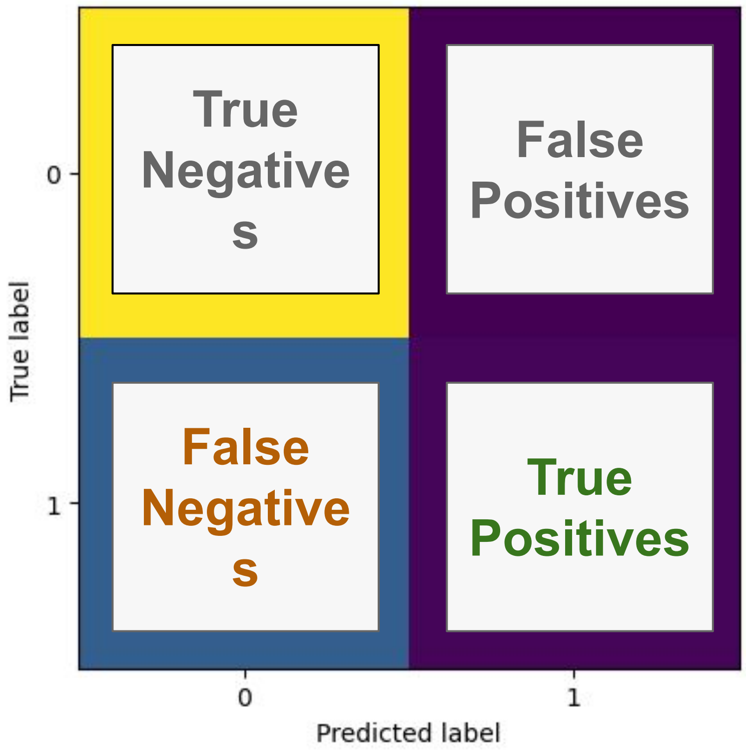
$$P = \frac{\boxed{TP}}{\boxed{TP} + \boxed{FP}} = 0.625$$

What proportion of positive ***identifications*** *was actually correct? 62.5%*

```
from sklearn.metrics import precision_score
precision_score(y_test, y_pred)
```

```
0.625
```

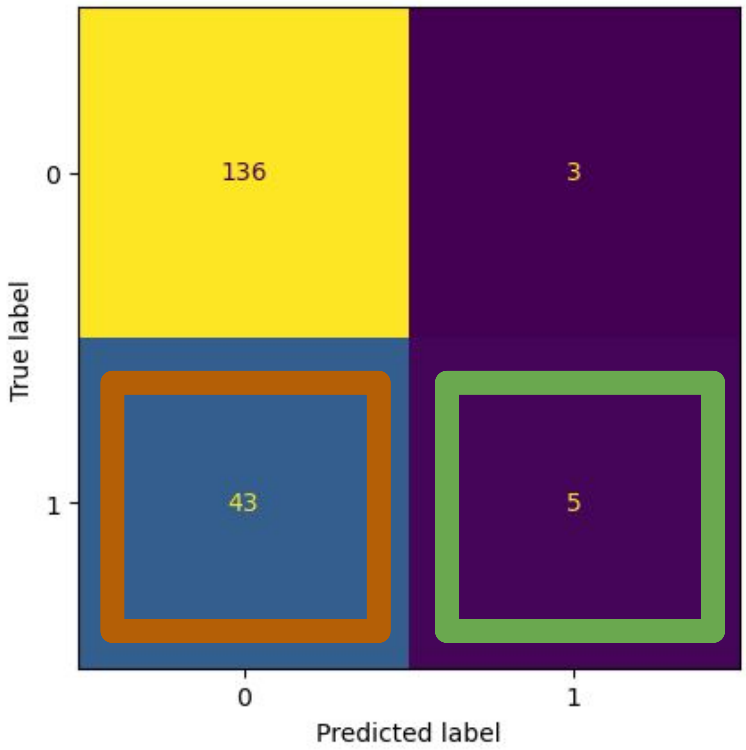
Recall



$$R = \frac{\boxed{TP}}{\boxed{TP} + \boxed{FN}}$$

*What proportion of **actual** positives was identified correctly?*

Recall



$$R = \frac{\boxed{TP}}{\boxed{TP} + \boxed{FN}} = 0.104$$

*What proportion of **actual** positives was identified correctly? 10.4%*

```
from sklearn.metrics import recall_score
recall_score(y_test, y_pred)
```

```
0.10416666666666667
```

F1-score

- What if we want a **balanced mix** of Precision and Recall?
- We can use the **harmonic mean** of the two metrics

$$F1 = 2\frac{precision \times recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

```python
from sklearn.metrics import f1_score
f1_score(y_test, y_pred)
```

```
0.17857142857142856
```

When to use them?

- We should look at **Precision** when the **cost of false positive is high**
  - E.g. when a false positive for an illness means someone will have to go through painful and invasive testing
- We should look at **Recall** when the **cost of false negative is high**
  - E.g. when a false negative could results in significant costs or bodily harm to someone
- We should look at **F1-score** when we want to consider false negatives and false positives equally.

8. Threshold Dependence

True Positive Rate & False Positive Rate

- **True Positive Rate (TPR)** is the same as **Recall**

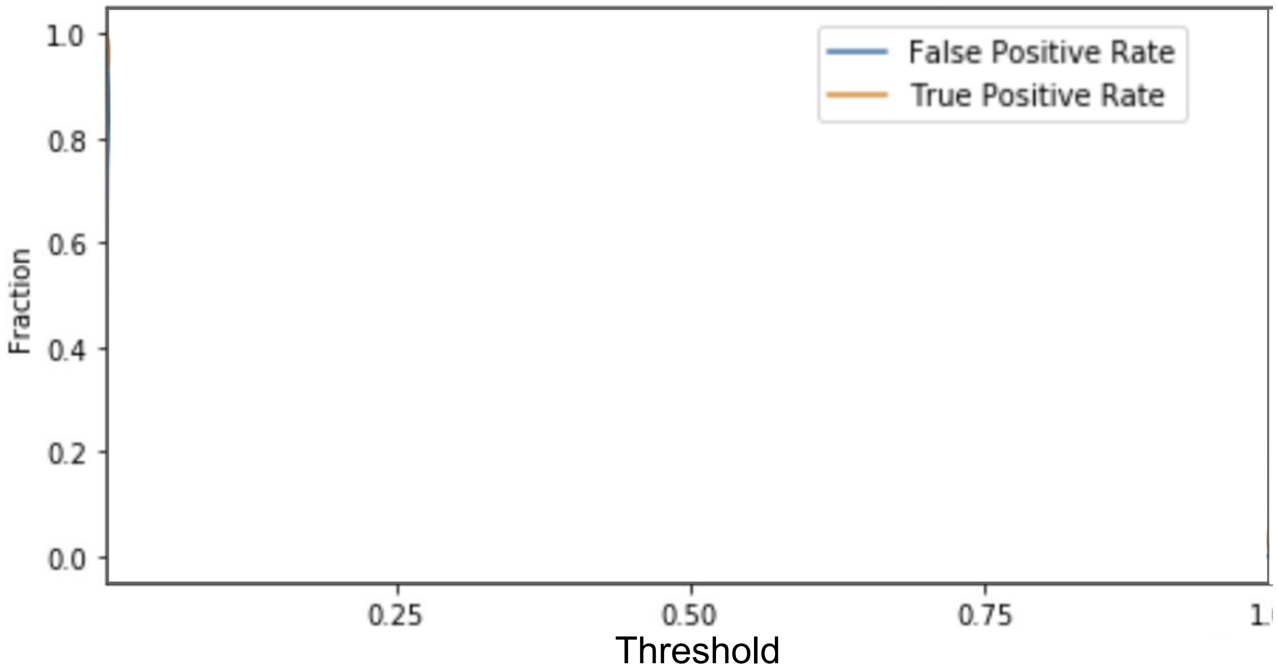
$$TPR = \frac{TP}{TP + FN}$$

- **False Positive Rate (FPR)** is the ratio of False Positives and the total Negatives

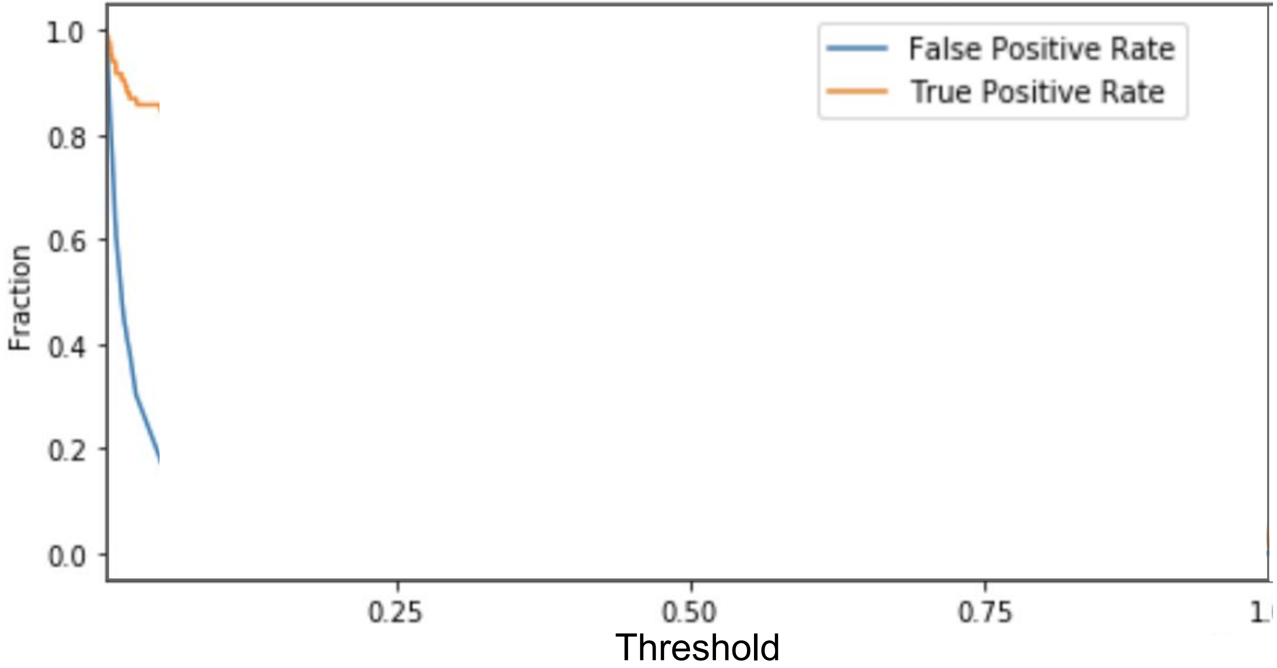
$$FPR = \frac{FP}{FP + TN}$$

- What happens to these metrics when we **change our threshold**?

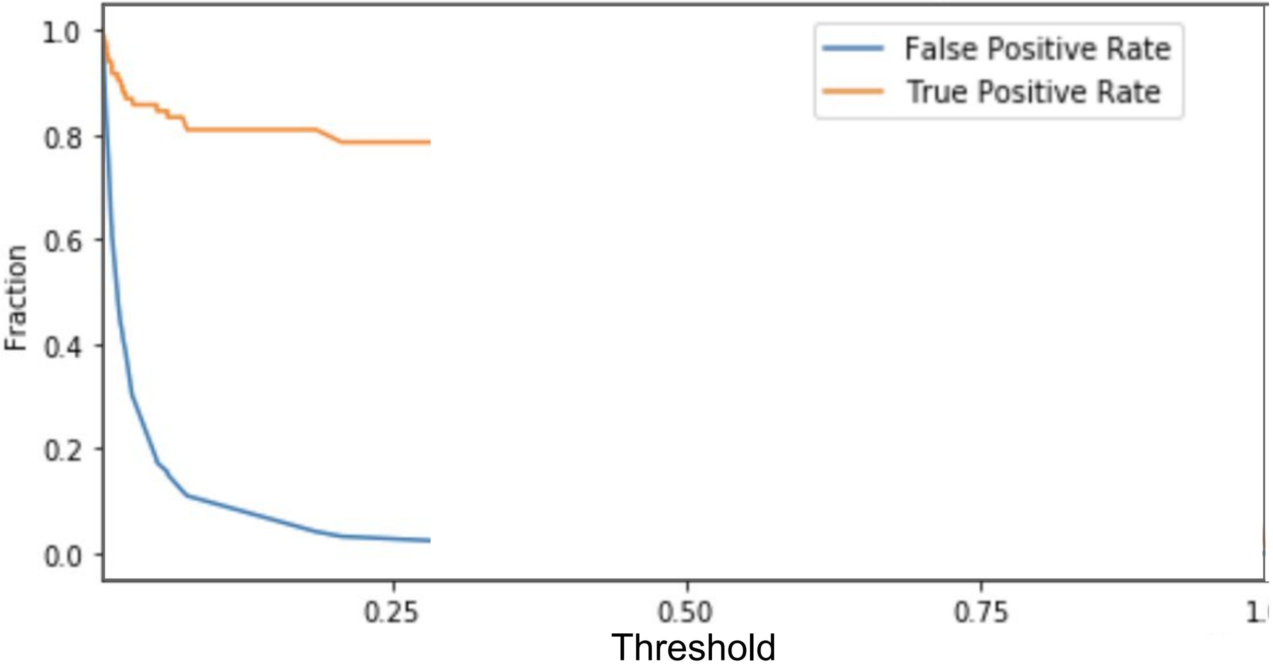
True Positive Rate & False Positive Rate



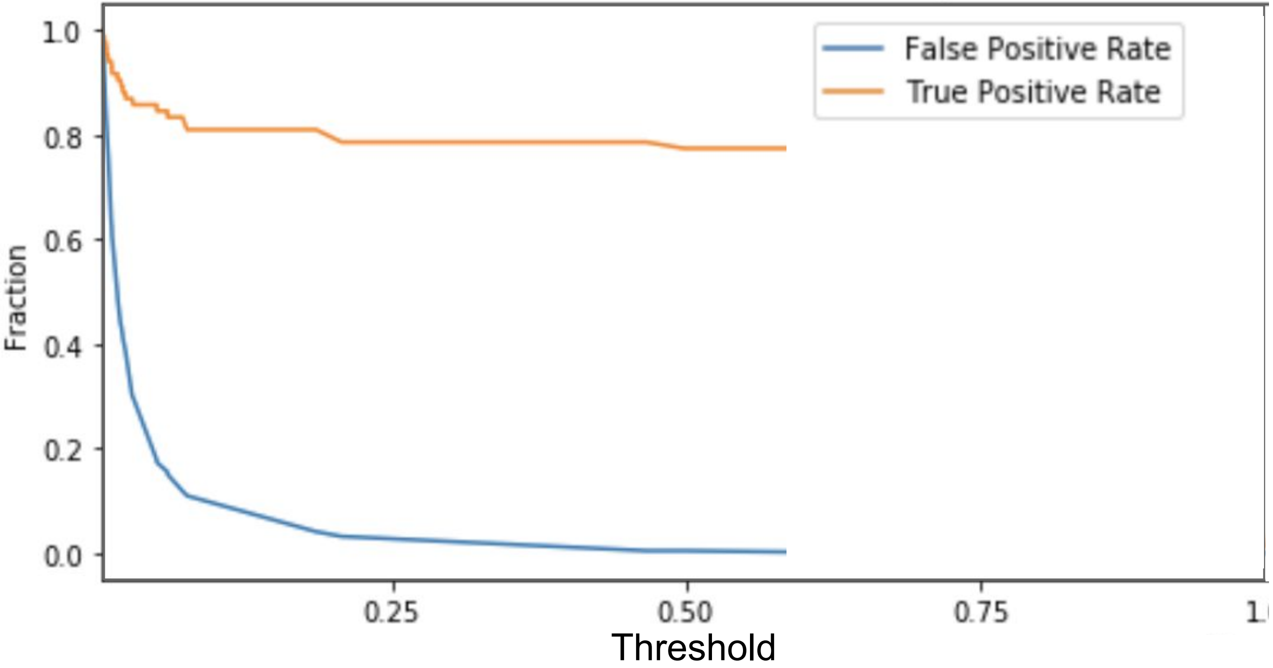
True Positive Rate & False Positive Rate



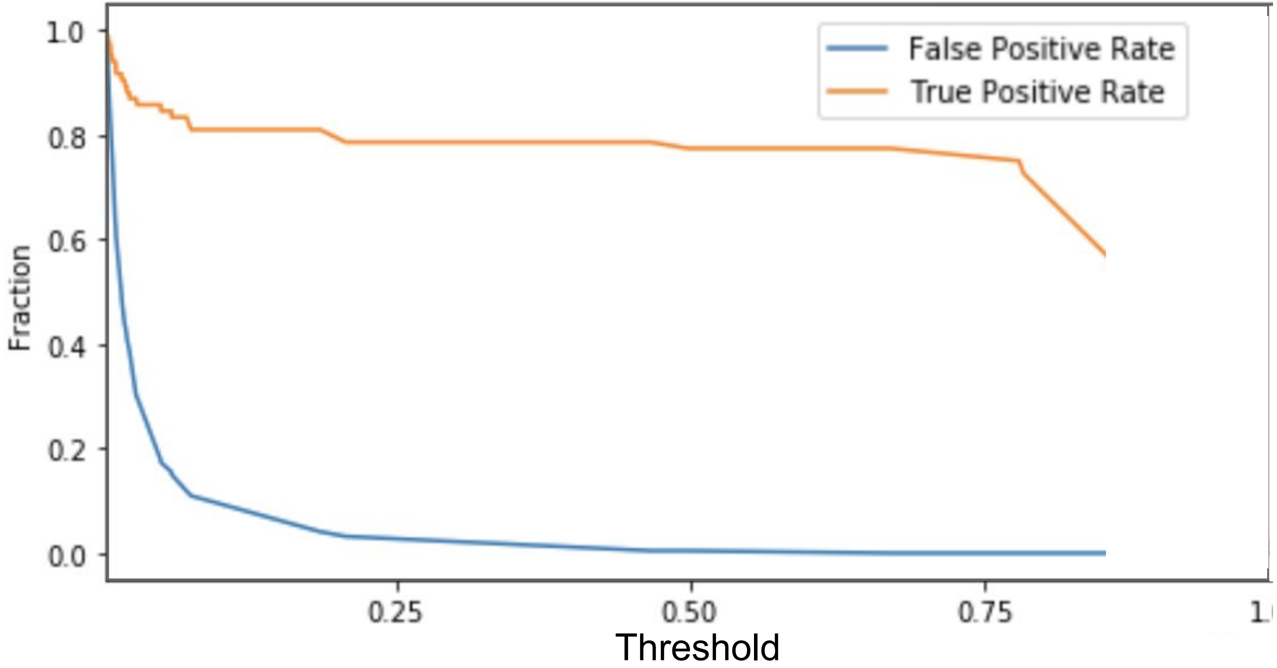
True Positive Rate & False Positive Rate



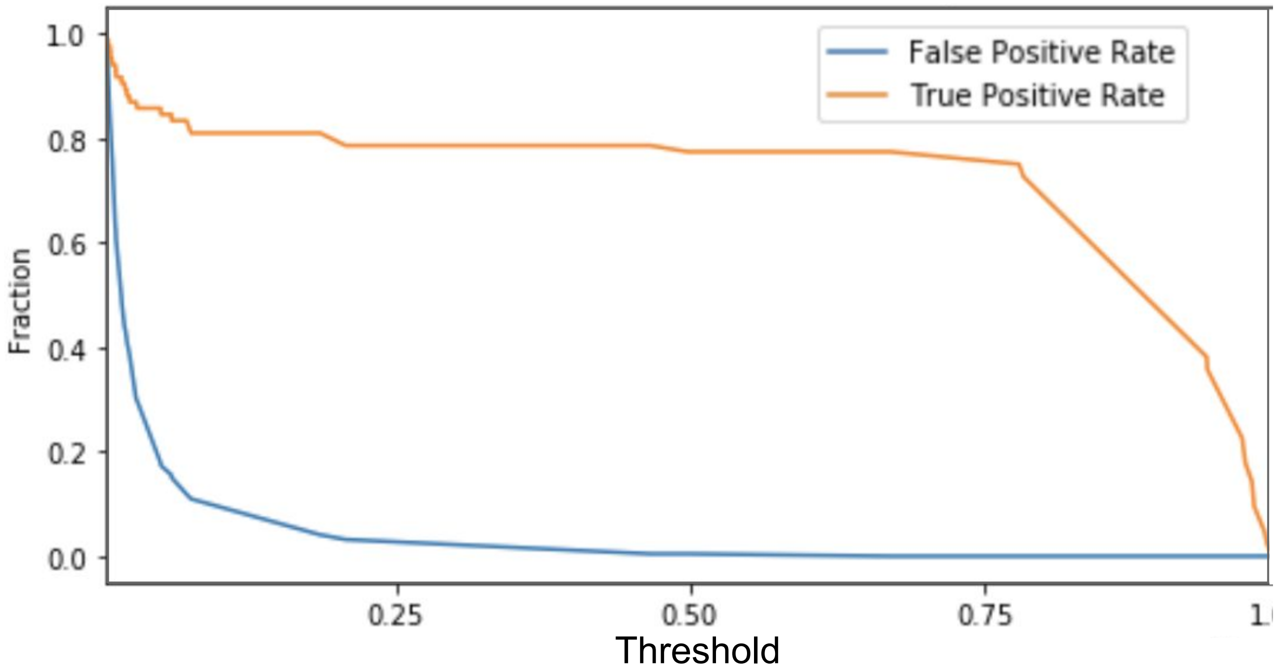
True Positive Rate & False Positive Rate



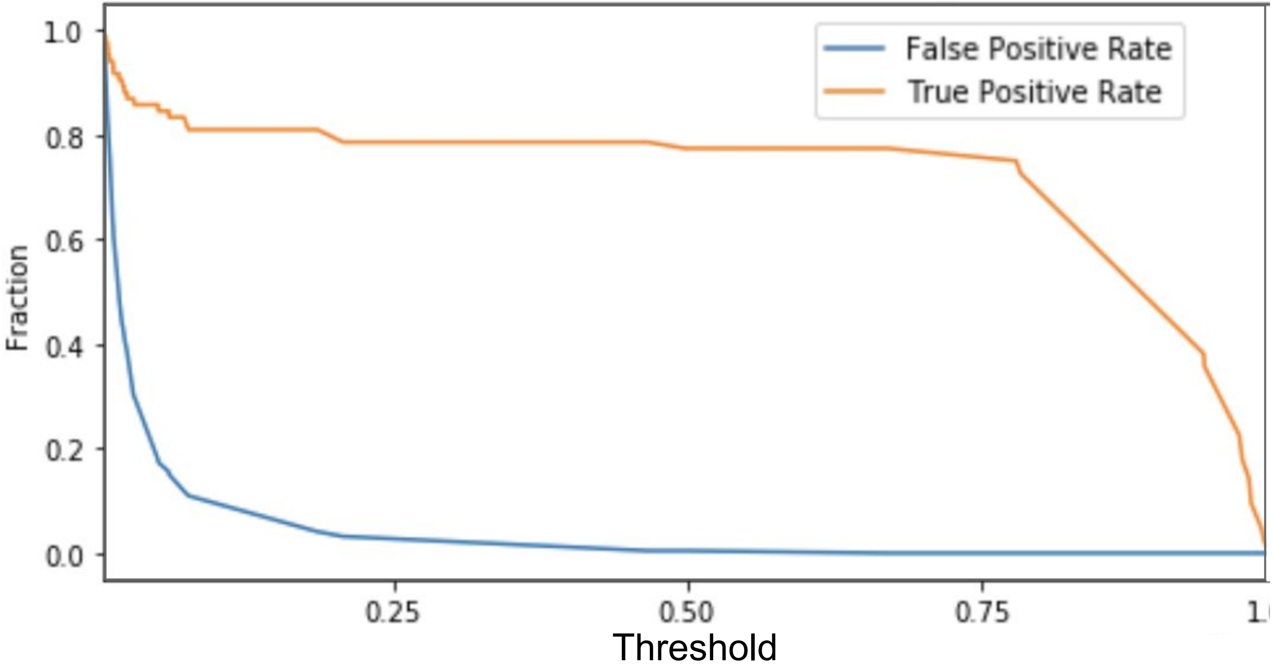
True Positive Rate & False Positive Rate



True Positive Rate & False Positive Rate



True Positive Rate & False Positive Rate

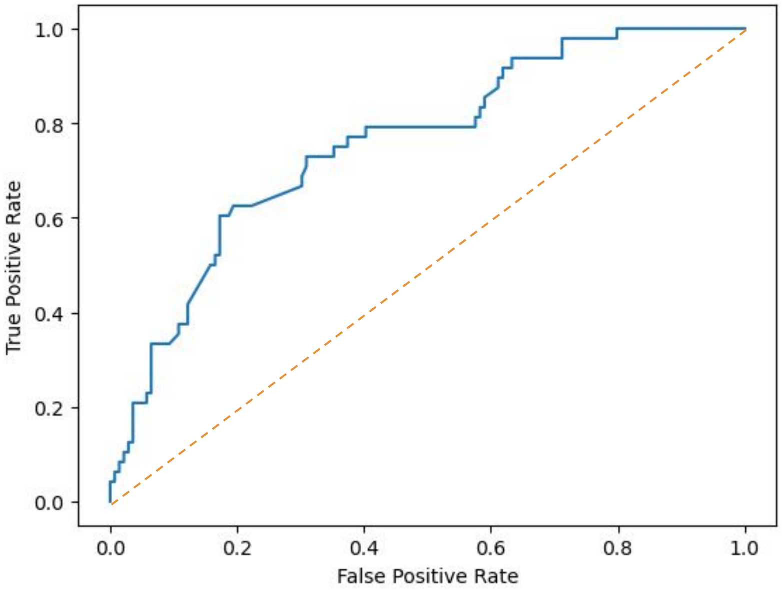


- What if we plot one against the other?

9. ROC and PR curves

The ROC curve

- The **Receiver Operating Characteristic (ROC) curve** is the relation between **TPR** and **FPR**
- The **Area Under the Curve (AUC)** for the ROC curve is a **threshold-independent metric**

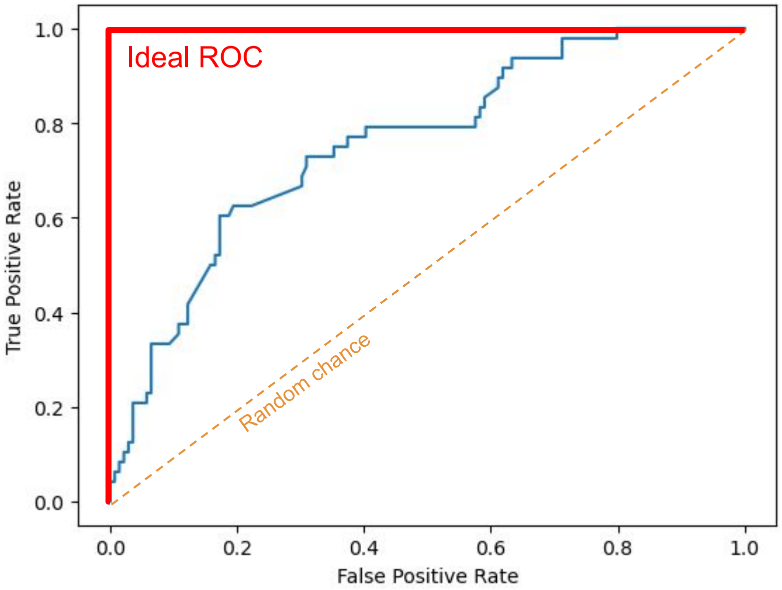



```python
from sklearn.metrics import (
    roc_curve,
    RocCurveDisplay,
    roc_auc_score,
)
y_prob = logistic_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
roc_auc_score(y_test, y_prob)
```

```
0.7569694244604317
```

The ROC curve

- The **Receiver Operating Characteristic (ROC) curve** is the relation between **TPR** and **FPR**
- The **Area Under the Curve (AUC)** for the ROC curve is a **threshold-independent metric**

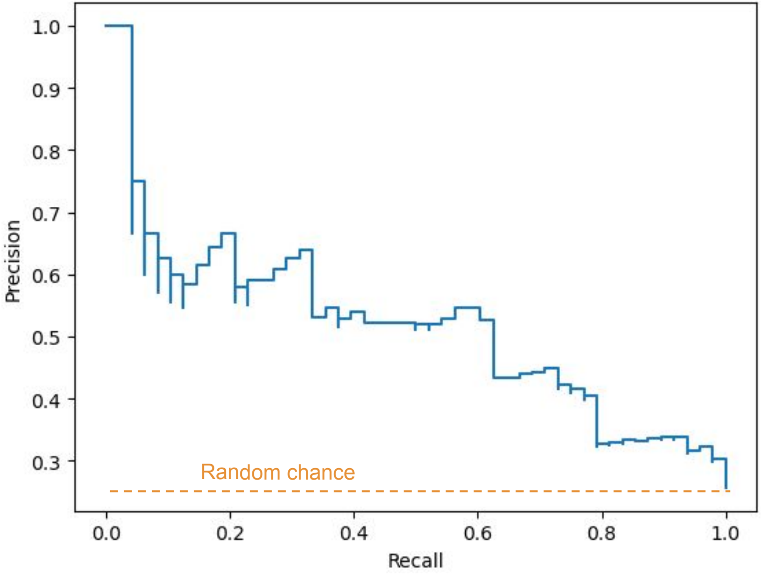



```python
from sklearn.metrics import (
    roc_curve,
    RocCurveDisplay,
    roc_auc_score,
)
y_prob = logistic_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
roc_auc_score(y_test, y_prob)
```

```
0.7569694244604317
```

Precision-Recall Curve

- What if we do the same for the **precision** and **recall**?
- The **Area Under the Curve (AUC)** for the **PR** curve is the **Average Precision**
  - It is also a **threshold-independent metric**




```python
from sklearn.metrics import (
    precision_recall_curve,
    PrecisionRecallDisplay,
    average_precision_score,
)
y_prob = logistic_model.predict_proba(X_test_scaled)[:,1]
prec, recall, _ = precision_recall_curve(y_test, y_prob)
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
average_precision_score(y_test, y_prob)
```

```
0.5192093401835707
```

Recap

- **Accuracy doesn’t show the whole picture**
  - E.g. in unbalanced and multiclass data sets

- **Precision** and **Recall** can give us a deeper insight into our performance, especially when we care a lot about False Positives or False Negatives

- **F1-score** tries to balance Precision and Recall

- **ROC curve** and its **AUC** are **threshold-independent** metrics

- **Precision-Recall curve** and **Average Precision** are also **threshold-independent**
  - The base random-chance line **depends** on the data set **base rate**

10. Q & A