# Topic 8 - Metrics for Performance Evaluation

# Aims of the Session

# Aims of the Session

- Learn different metrics used to evaluate classification frameworks

# Aims of the Session

- Learn different metrics used to evaluate classification frameworks
- Understand some alternatives to design proper tests

# Resources for the Lecture

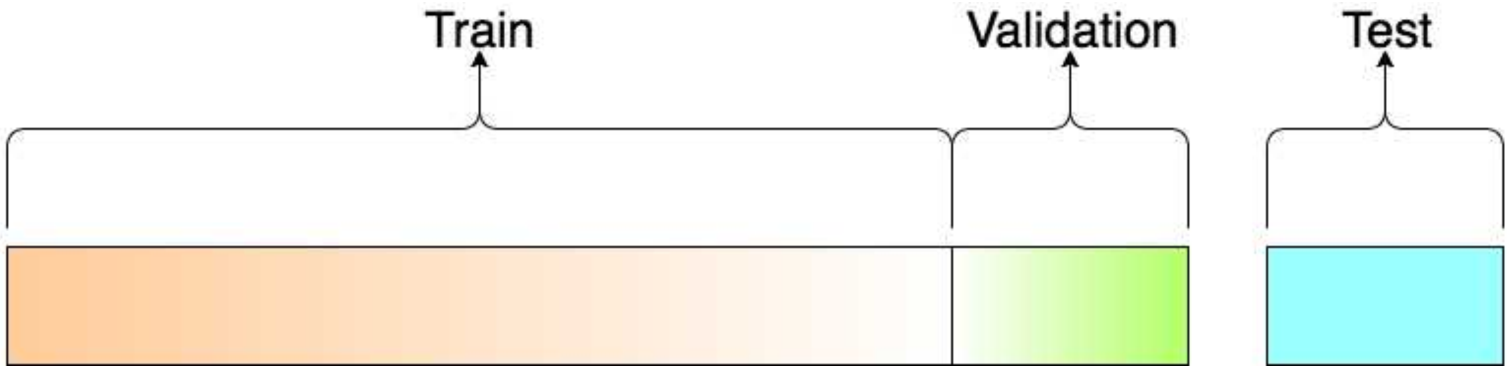
## Websites


- https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
- https://en.wikipedia.org/wiki/Precision_and_recall
- https://en.wikipedia.org/wiki/Sensitivity_and_specificity
- https://en.wikipedia.org/wiki/Confusion_matrix
- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://machinelearningmastery.com/k-fold-cross-validation/
- https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
- https://medium.com/mlearning-ai/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f

## Online Courses

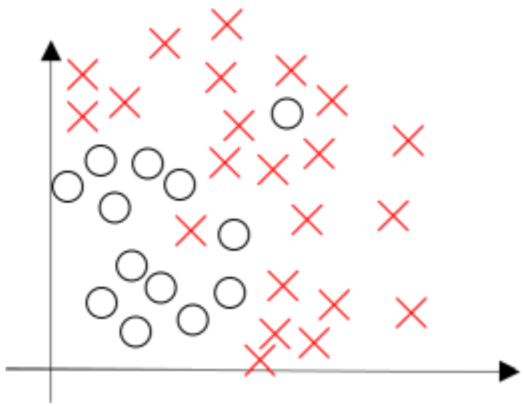
- Deep Learning Specialization by Andrew NG (Coursera)

# Some important concepts


Train
Validation
Test

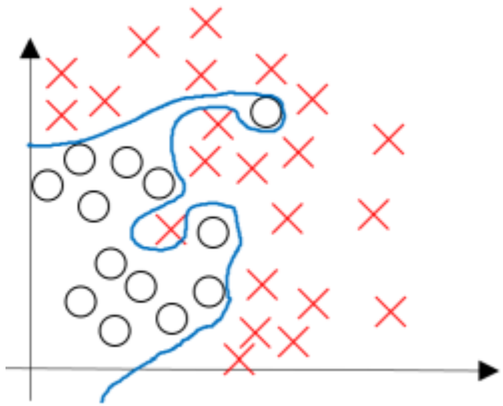
- `Generalisation` : The ability to correctly classify new examples different from those used for training a model

- `Generalisation` : The ability to correctly classify new examples different from those used for training a model



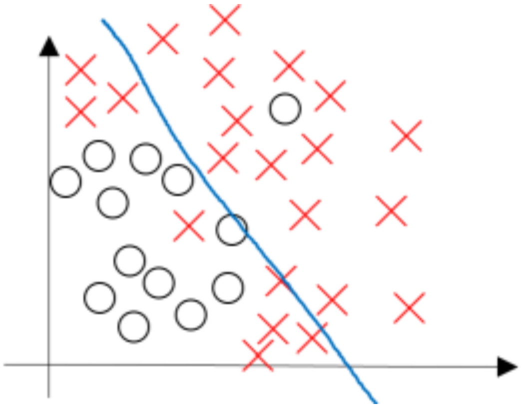
- `Overfitting` : The trained classifier gets a $100\%$ accuracy in the training/validation data, but only $50\%$ in the testing data.
    - Also known as `high variance` .

- `Overfitting` : The trained classifier gets a $100\%$ accuracy in the training/validation data, but only $50\%$ in the testing data.
    - Also known as `high variance` .



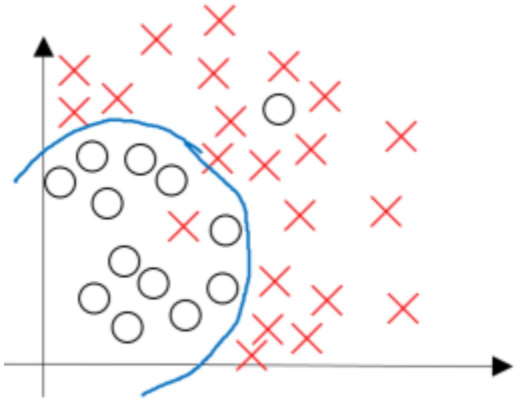
- `Underfitting` : The learned classifier is so simplistic that does not capture the structure of the data.
    - This translates on a poor performance on the validation data
    - Also known as `high bias`

- `Underfitting` : The learned classifier is so simplistic that does not capture the structure of the data.
    - This translates on a poor performance on the validation data
    - Also known as `high bias`



- What do we expect?

- What do we expect?



“just right”

# The bias-variance trade-off

# The bias-variance trade-off

- As you can see, a model can either have high bias or high variance

# The bias-variance trade-off

- As you can see, a model can either have high bias or high variance
- The main objective of machine learning is to find a function $h(x)$ that maps feature $X$ to class/target $y$ minimising:
    - bias error
    - variance error
    - irreducible error (noise in the data)

Typically[1], the **Error** of a learner/classifier is modelled using the following equation:

Typically[1], the **Error** of a learner/classifier is modelled using the following equation:

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

Typically[1], the **Error** of a learner/classifier is modelled using the following equation:

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

**Why $Bias^2$?**

# Performance Measures

# Performance Measures

- Assume that we are evaluating the classification success of a **binary** dataset

# Performance Measures

- Assume that we are evaluating the classification success of a **binary** dataset
- `True Positives` (TP): This is what many people understand as *accuracy* (but is not!)
    - Samples from the *positive class* that are classified correctly

# Performance Measures

- Assume that we are evaluating the classification success of a **binary** dataset
- `True Positives` (TP): This is what many people understand as *accuracy* (but is not!)
    - Samples from the *positive class* that are classified correctly
- `True Negatives` (TN): How many samples from the negative class are **NOT** classified as being from the positive one

- `False Positives` (FP): How many samples from the negative class are classified as being from the positive class
    - Also known in statistics as **False alarms** or **Type I Error**

- `False Positives` (FP): How many samples from the negative class are classified as being from the positive class
    - Also known in statistics as **False alarms** or **Type I Error**

- `False Negatives` (FN): How many samples from the positive class are classified as being from the negative class
    - Also known in statistics as **Type II Error**

# Accuracy

# Accuracy

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

# Accuracy

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- The value of the accuracy must be **between** $0$ **and** $1$

# Accuracy

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- The value of the accuracy must be **between** $0$ **and** $1$
- Recall that we said that this is **not** a good measure for imbalanced datasets

# Accuracy

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

- The value of the accuracy must be **between** $0$ **and** $1$

- Recall that we said that this is **not** a good measure for imbalanced datasets

- **WHY?**

# Error Rate

## Error Rate

- $Error\ Rate = \frac{FP+FN}{TP+TN+FP+FN} = 1 - Accuracy$

# Error Rate

- $Error\ Rate = \frac{FP+FN}{TP+TN+FP+FN} = 1 - Accuracy$
- Also must be **between** $0$ **and** $1$

# Error Rate

- $Error\ Rate = \frac{FP+FN}{TP+TN+FP+FN} = 1 - Accuracy$
- Also must be **between** $0$ **and** $1$
- **Do you think this one is good for imbalanced datasets?**

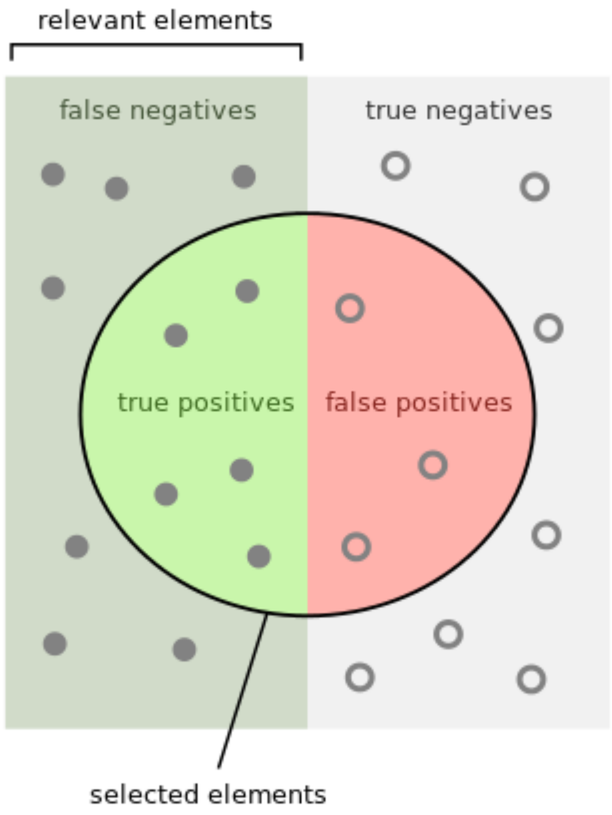
# Precision and Recall

# Precision and Recall

- Assume that we have the following **binary** classification scenario

# Precision and Recall

- Assume that we have the following **binary** classification scenario



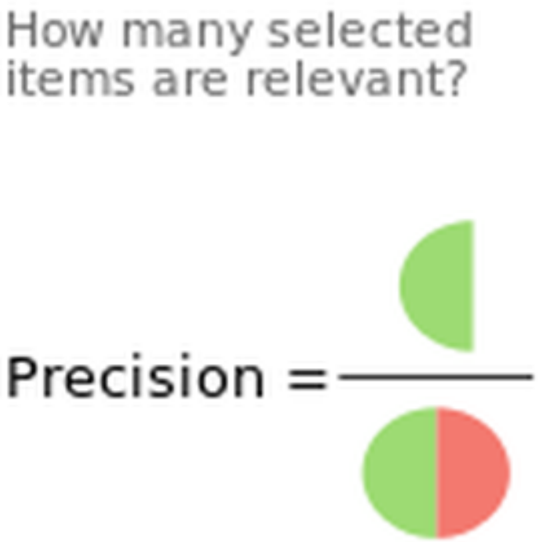
Precision

## Precision

- $Precision = \frac{TP}{TP+FP}$

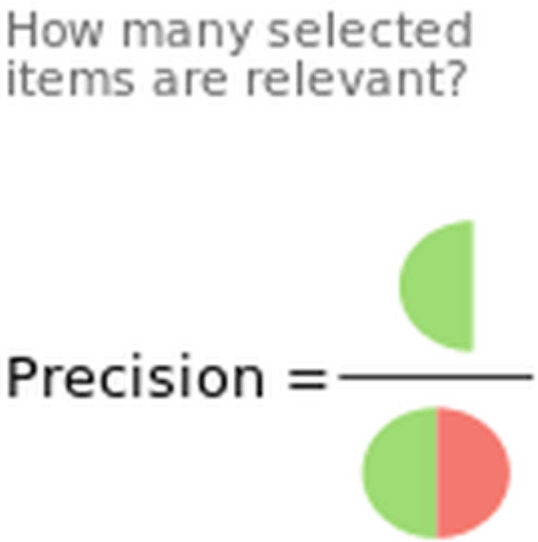
## Precision

- $Precision = \frac{TP}{TP+FP}$



Precision

- $Precision = \frac{TP}{TP+FP}$




- How much of what I **have** I **need**?

Recall

## Recall

- $Recall = \frac{TP}{TP+FN}$

Recall

- $Recall = \frac{TP}{TP+FN}$

Recall

- $Recall = \frac{TP}{TP+FN}$


- How much of what I **need** I **have**?

- The difference is in what you divide the `TP` with

- The difference is in what you divide the `TP` with
- Most systems are known to have a precision/recall trade-off

- The difference is in what you divide the `TP` with
- Most systems are known to have a precision/recall trade-off
- **Which is better?**

F1-score (or F1-measure)

F1-score (or F1-measure)

- Harmonic mean between precision and recall

F1-score (or F1-measure)

- Harmonic mean between precision and recall
- $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{(2 \times TP) + FP + FN}$

# Sensitivity and Specificity

## Sensitivity and Specificity

- Similar to precision and recall, but used more in the health sciences domain

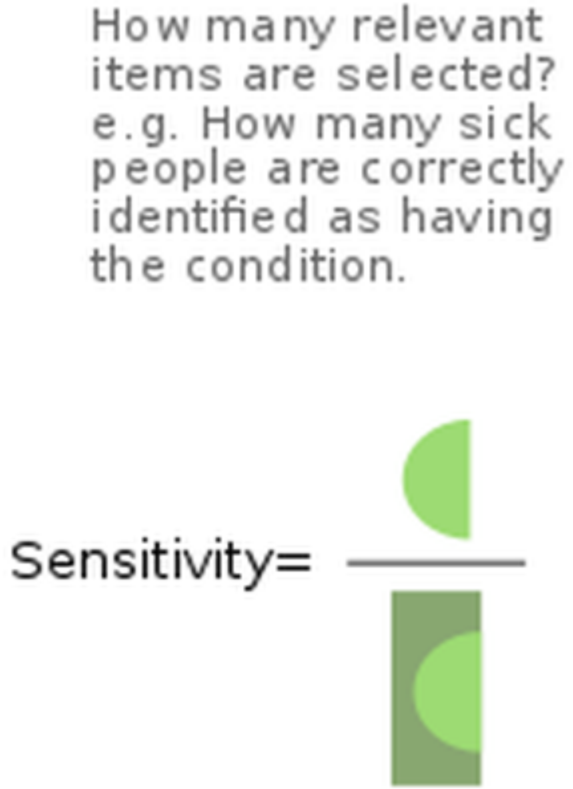
# Sensitivity

# Sensitivity

- Just another name for **recall**

## Sensitivity

- Just another name for **recall**



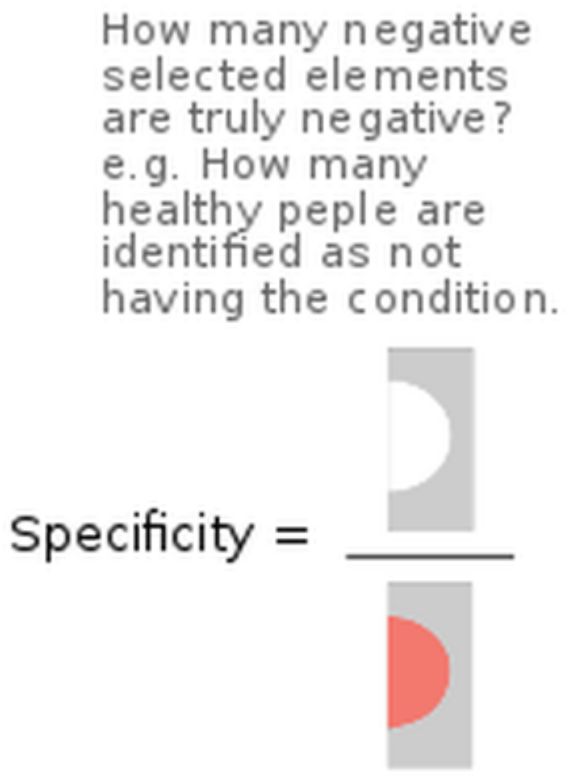
# Specificity

## Specificity

- The precision for the negative class

Specificity

- The precision for the negative class



**Is there any "F-measure" for these two?**

# The Confusion Matrix

# The Confusion Matrix

- Also known as *error matrix*

# The Confusion Matrix

- Also known as *error matrix*
- Table that allows you to visualise the performance of a supervised learning algorithm

# The Confusion Matrix

- Also known as *error matrix*
- Table that allows you to visualise the performance of a supervised learning algorithm

Example

# The Confusion Matrix

- Also known as *error matrix*
- Table that allows you to visualise the performance of a supervised learning algorithm

## Example

- A classifier has been trained to distinguish cats from dogs

- Assuming a sample of 13 animals (8 cats and 5 dogs), you get the following confusion matrix

- Assuming a sample of 13 animals (8 cats and 5 dogs), you get the following confusion matrix

| | | Actual class | |
|---|---|---|---|
| | | **Cat** | **Dog** |
| **Predicted class** | **Cat** | **5** | 2 |
| | **Dog** | 3 | **3** |

- This table can also be interpreted with respect to the previously seen terms

- This table can also be interpreted with respect to the previously seen terms

| | | Actual class | |
|---|---|---|---|
| | | **Cat** | **Non-cat** |
| **Predicted class** | **Cat** | 5 True Positives | 2 False Positives |
| | **Non-cat** | 3 False Negatives | 3 True Negatives |

Area under the Receiving Operating Characteristic (ROC) Curve

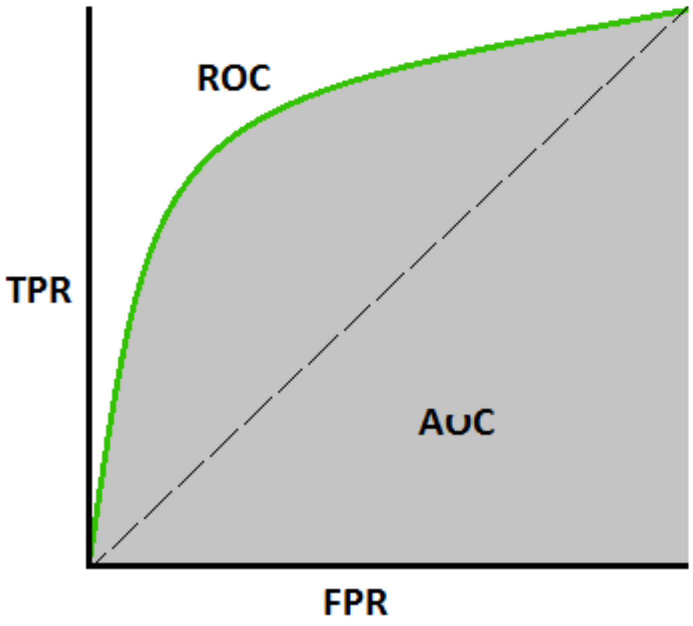
- Suitable to compare classification rates in a more visual way and at **different threshold settings**

- Suitable to compare classification rates in a more visual way and at **different threshold settings**
- It is a probability curve that tells you how much your model is able to distinguish between classes

- Suitable to compare classification rates in a more visual way and at **different threshold settings**
- It is a probability curve that tells you how much your model is able to distinguish between classes
- Higher the AUC, better the model is capable of performing the distinction

- The curve plots **False Positive Rate** (x-axis) vs **True Positive Rate** (y-axis)
    - $FPR : 1 - Specificity$
    - $TPR : Recall\ (also\ known\ as\ Sensitivity)$

- The curve plots **False Positive Rate** (x-axis) vs **True Positive Rate** (y-axis)
    - $FPR : 1 - Specificity$
    - $TPR : Recall \ (also \ known \ as \ Sensitivity)$



# Runtime

# Runtime

- It's not a bad idea to report this, particularly in large image datasets

## Runtime

- It's not a bad idea to report this, particularly in large image datasets
- Not very "accepted" in the academic world, but extremely useful in the industrial one!

# Runtime

- It's not a bad idea to report this, particularly in large image datasets
- Not very "accepted" in the academic world, but extremely useful in the industrial one!
- You can import the `time` module in Python and use the `perf_counter()` function to calculate the time of processes running
    - Just be very careful where in your code you calculate the time!

In [1]:

```
import time

t = time.perf_counter()
# do stuff
x=0
for i in range(1000):
    x=x+i
# stuff has finished
print('Elapsed time: ',time.perf_counter() - t)
```

```
Elapsed time:  0.00035020000007079943
```

# What about multi-class classification?

# What about multi-class classification?

- So far, we have only spoken of metrics in the context of binary datasets

# What about multi-class classification?

- So far, we have only spoken of metrics in the context of binary datasets
- However, in most cases you will deal with multi-class datasets

# What about multi-class classification?

- So far, we have only spoken of metrics in the context of binary datasets
- However, in most cases you will deal with multi-class datasets
- There are many ways to adapt the aforementioned metrics to these scenarios, the most common one being the **One vs All** approach
    - Comparing a metric of one class against the rest as if these were a single class

- Considering that you can still calculate precision, recall and F1-score for each class (against the rest), another commonly used approach is **macro/weighted/micro** metrics:

- Considering that you can still calculate precision, recall and F1-score for each class (against the rest), another commonly used approach is **macro/weighted/micro** metrics:

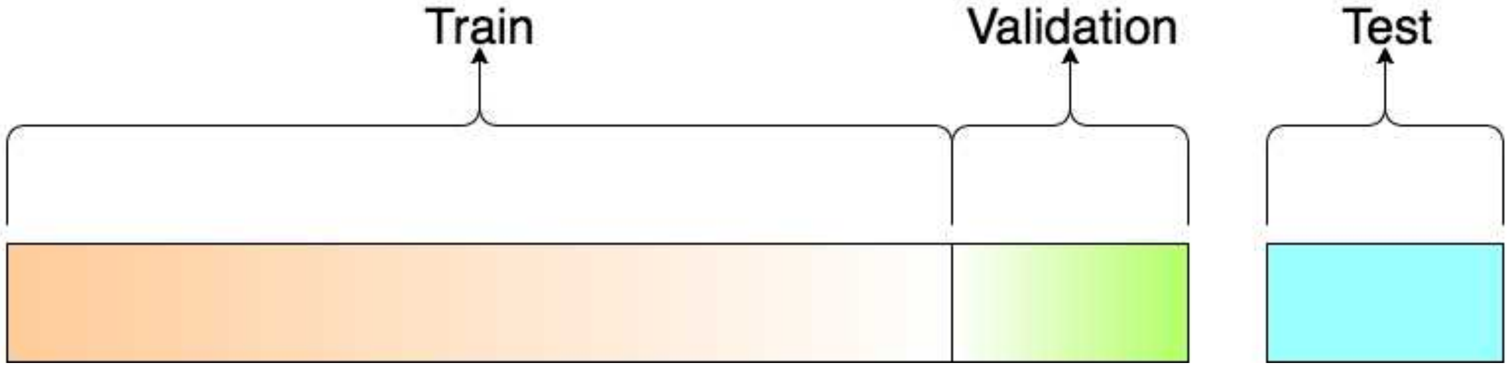
- `Macro` is the arithmetic mean of all metrics

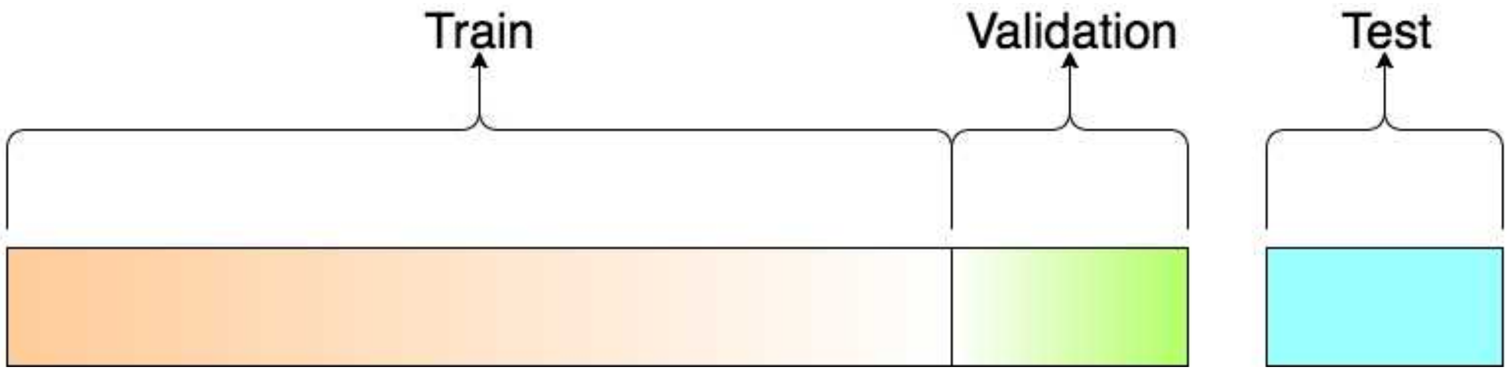
- Considering that you can still calculate precision, recall and F1-score for each class (against the rest), another commonly used approach is **macro/weighted/micro** metrics:
- `Macro` is the arithmetic mean of all metrics
- `Weighted` is when we multiply each metric by the number of samples of each class

- Considering that you can still calculate precision, recall and F1-score for each class (against the rest), another commonly used approach is **macro/weighted/micro** metrics:
- `Macro` is the arithmetic mean of all metrics
- `Weighted` is when we multiply each metric by the number of samples of each class
- `Micro` is the harmonic mean of all metrics, which derives in the system's accuracy

- Considering that you can still calculate precision, recall and F1-score for each class (against the rest), another commonly used approach is **macro/weighted/micro** metrics:
- `Macro` is the arithmetic mean of all metrics
- `Weighted` is when we multiply each metric by the number of samples of each class
- `Micro` is the harmonic mean of all metrics, which derives in the system's accuracy
- To see an example of this, I recommend you to visit this site

# Validation Frameworks


Train
Validation
Test




- Technically this is not the only way to split the data!

- Even when you split uniformly using train/val/test approach, you are still not considering that maybe some train/val data is better/worse for testing and vice versa!

- Even when you split uniformly using train/val/test approach, you are still not considering that maybe some train/val data is better/worse for testing and vice versa!
- To address this issue, there are some iterative validation frameworks which let you split data in different ways and perform multiple tests of the same model

# Cross validation

# Cross validation

- Simple to understand

# Cross validation

- Simple to understand
- Reduces "bias"
  - i.e. over-optimistic results that may be caused due to chance

# Cross validation

- Simple to understand
- Reduces "bias"
    - i.e. over-optimistic results that may be caused due to chance
- Based on a single parameter $k$ which defined the number of times that the dataset will be *folded*

# How it works

## How it works

1. Shuffle the dataset

## How it works

1. Shuffle the dataset
2. Split the dataset into $k$ groups

3. For each group
    - Take that group as the test data
    - Take the remaining groups as the training data
    - Fit the model
    - Retain the score and discard the model

3. For each group
    - Take that group as the test data
    - Take the remaining groups as the training data
    - Fit the model
    - Retain the score and discard the model

4. Once you are done, average/summarise all results

Which $k$ to choose?

Which $k$ to choose?

- Representative for the model: Large enough to be statistically significant!

Which $k$ to choose?

- Representative for the model: Large enough to be statistically significant!
- $k = 5$ and $k = 10$ are the usual standard, but it depends on how many samples you have!

- If you do $k = n$ ($n$ being the number of samples in the dataset) then you will test every sample as the test against the rest as the training set

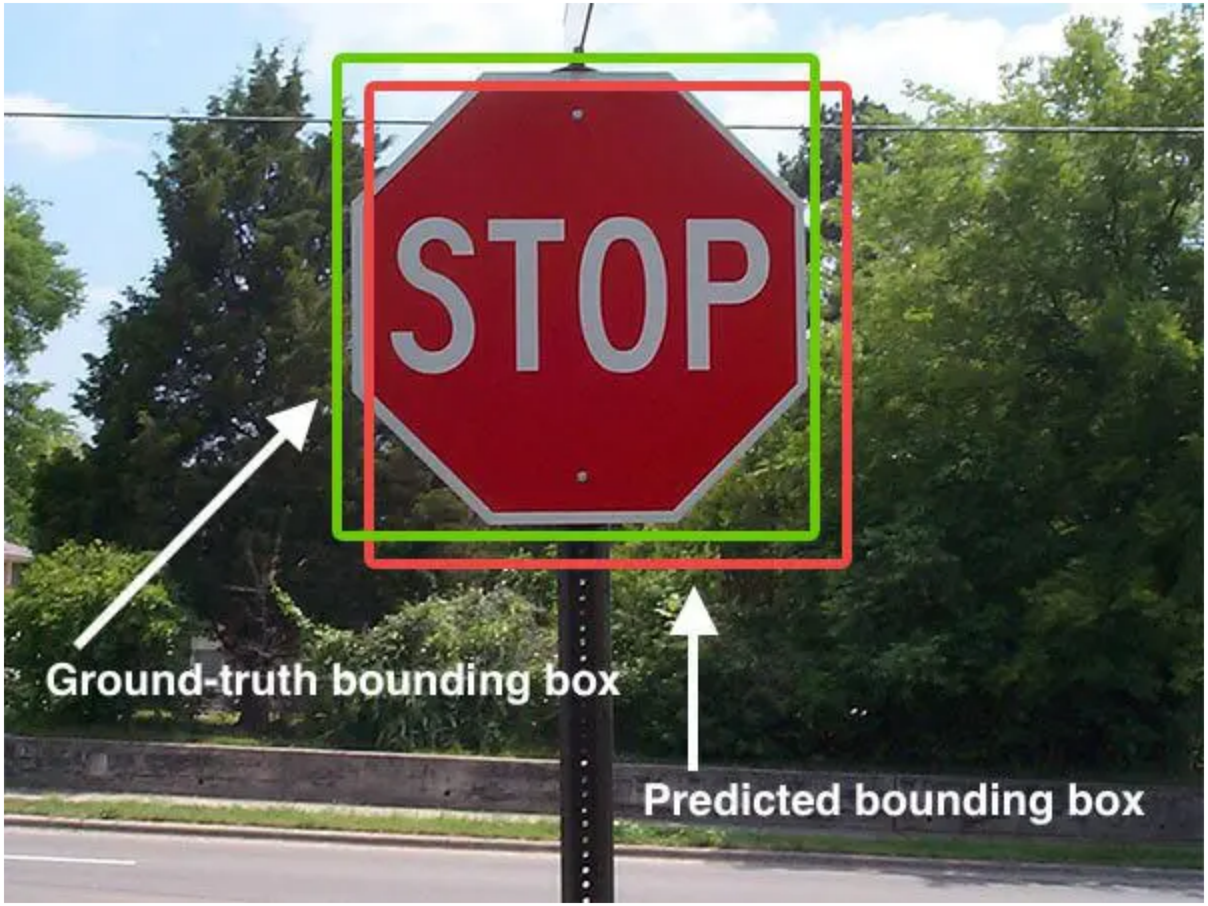
- If you do $k = n$ ($n$ being the number of samples in the dataset) then you will test every sample as the test against the rest as the training set
- This is also known as the **Leave-One-Out** approach

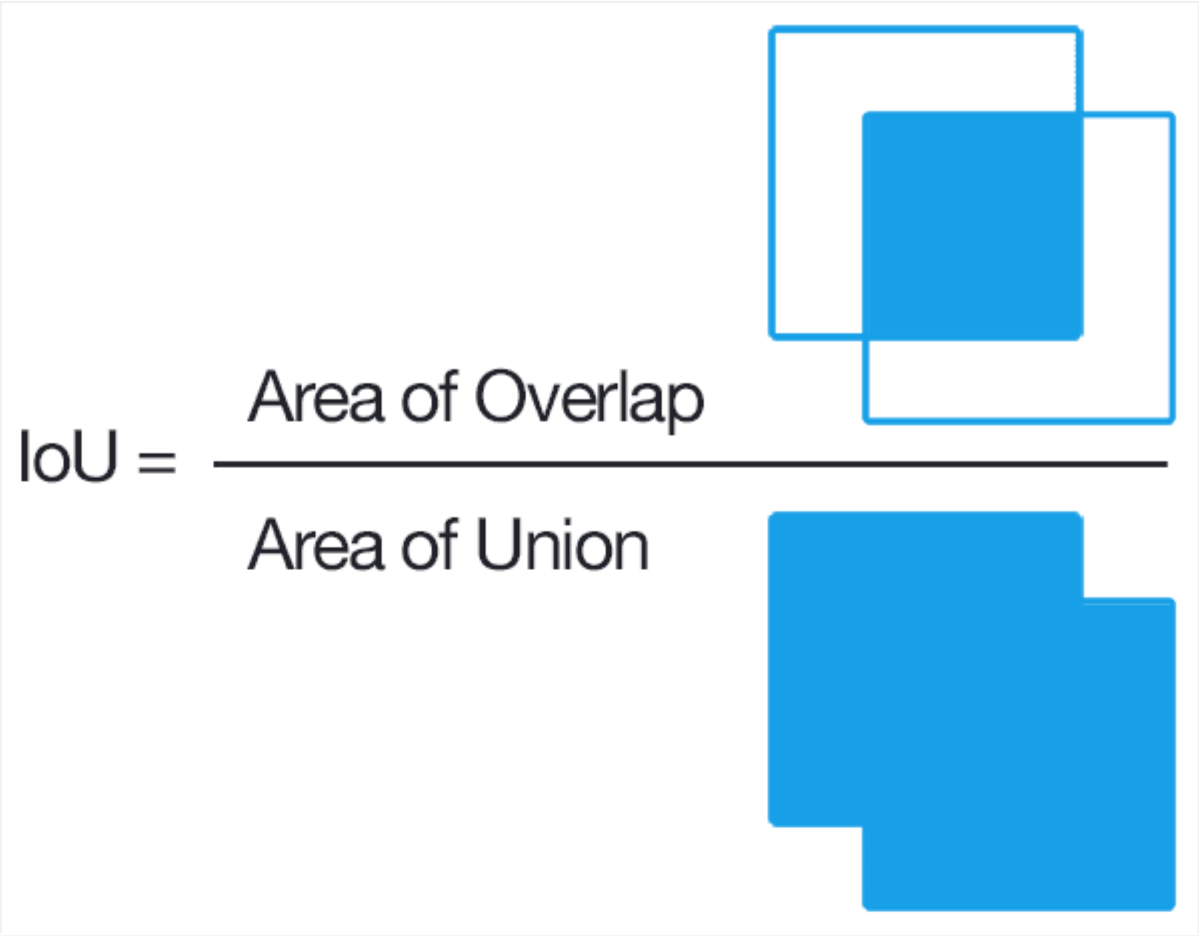
- Some datasets (like the one you will use in the bonus part of the lab) already are partitioned in the $k$ folds

# Metrics used in Computer Vision

# IoU (A.K.A. Jaccard Index)

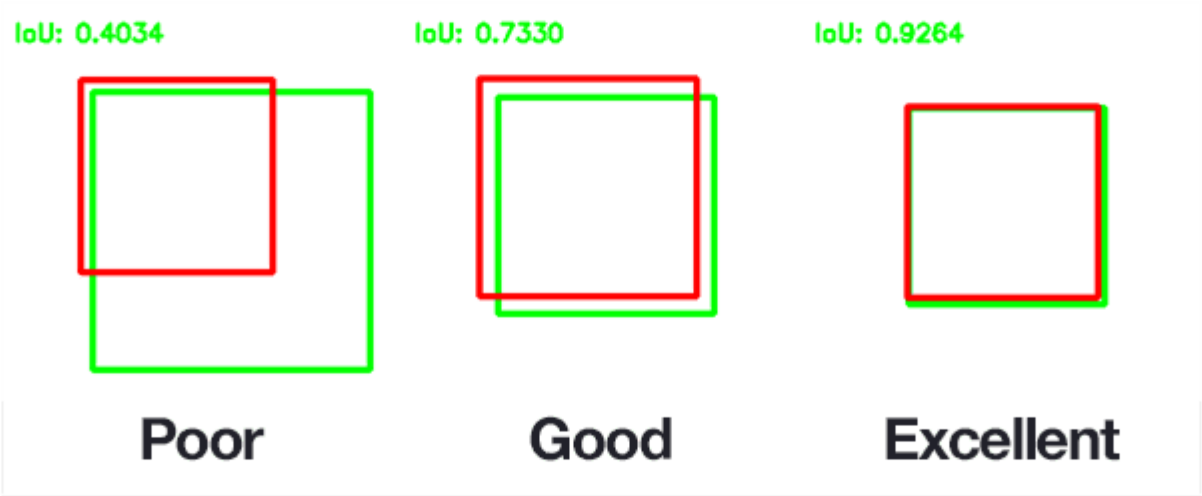
# IoU (A.K.A. Jaccard Index)




Area of Overlap
IoU =
Area of Union

- Normally, $IoU \geq 0.5$ is considered good, while $1$ is perfect!

- Normally, $IoU \geq 0.5$ is considered good, while $1$ is perfect!



# Dice Coefficient

# Dice Coefficient

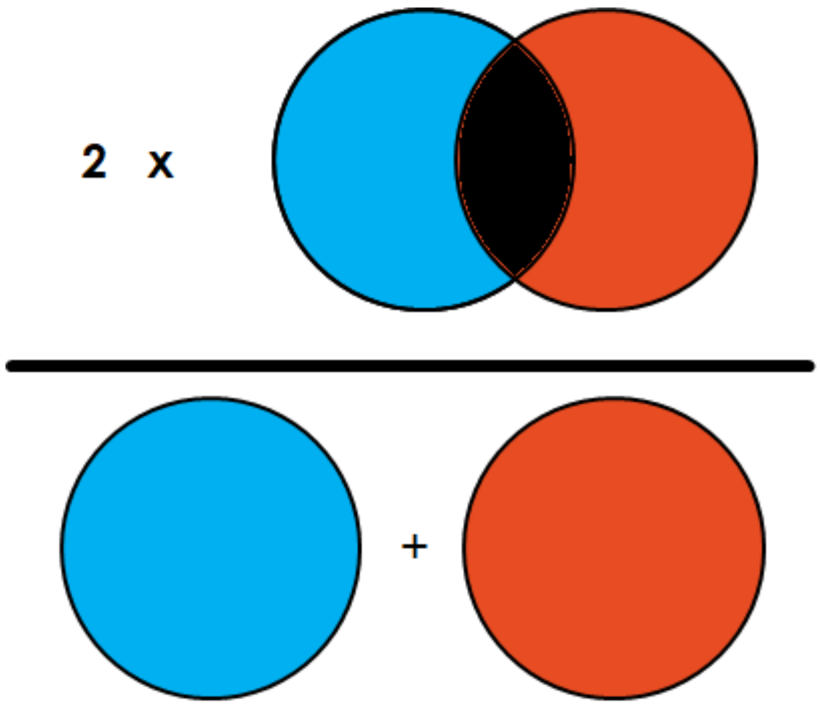
- The "F1-Score" of computer vision metrics

# Dice Coefficient

- The "F1-Score" of computer vision metrics
- More widely used for segmentation

# Dice Coefficient

- The "F1-Score" of computer vision metrics
- More widely used for segmentation



## What is the difference between IoU and Dice?

**What is the difference between IoU and Dice?**

- IoU is more like recall, so it is good to use when you want to detect if a larger amount of the object pixels are outside the area of interest, but also if the detection is **overestimating** where the object is!

**What is the difference between IoU and Dice?**

- IoU is more like recall, so it is good to use when you want to detect if a larger amount of the object pixels are outside the area of interest, but also if the detection is **overestimating** where the object is!
- Dice coefficient penalises false positives, which is better for high imbalanced datasets or when the segmentations are not correct

# LAB: PERFORMANCE MEASURES FOR BINARY DATASETS