



---

# Machine Learning (ML) with Python

## Model Evaluation

Dr. Aeshah Alsughayyir

Collage of Computer Science and Engineering

Taibah University

2021-2022

---

# Outline:

---

- **Model Evaluation Metrics**
  - Types of Predictive models
  - Confusion Matrix
    - Accuracy
    - Precision and Recall
    - F1 Score
  - Log Loss
  - Root Mean Squared Error
  - Cross Validation (Not a normal metric though!)
    - K-Fold Cross Validation

# Model Evaluation Metrics

# Types of Predictive models

---

- Talking about predictive models is talking either about a regression model (continuous output) or a classification model (nominal or binary output). **The evaluation metrics used in each of these models are different.**
- So, it is beneficial to know which evaluation metric will properly measure your model performance.
- Certain metrics measure model performance better than others, depending on the use case.

# Types of Predictive models (Cont.)

---

- **In classification problems**, we use two types of algorithms (dependent on the kind of output it creates):
  - **Class output:** Algorithms like *SVM* and *KNN* create a class output. For instance, in a binary classification problem, the outputs will be either 0 or 1. However, today we have algorithms which can convert these class outputs to probability. But these algorithms are not well accepted by the statistics community.
  - **Probability output:** Algorithms like *Logistic Regression*, *Random Forest*, *Gradient Boosting*, *Adaboost* etc. give probability outputs. Converting probability outputs to class output is just a matter of creating a threshold probability.
- **In regression problems**, we do not have such inconsistencies in output. The output is always continuous in nature and requires no further treatment.

We will explore some of the most common evaluation metrics for both regression and classification models.

# Quick Revision

---

- To evaluate any predictive (supervised) models we have to split our dataset into a training set and a test set.
- The splitting process in Python is done using the `train_test_split` function.

```
from sklearn.model_selection import train_test_split

# split data and labels into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

- built a model on the training set by calling the fit method and evaluated it on the test set using score method:

```
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)

# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))
```

# Quick Revision (Cont.)

---

- The reason we split our data into training and test sets is that:
  - We are interested in **measuring how well our model generalizes** to new, previously unseen data.
  - We are not interested in how well our model fit the training set, but rather in **how well it can make predictions for data that was not observed during training.**

**An evaluation metric** quantifies the performance of a predictive model over unseen data

# Confusion Matrix

A **confusion matrix** is a summary of prediction results on a classification problem.

The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

The **confusion matrix** shows the ways in which your classification model is *confused* when it makes predictions.

## How to Calculate a Confusion Matrix

The process for calculating a confusion Matrix.

1. You need a test dataset or a validation dataset with expected outcome values.
2. Make a prediction for each row in your test dataset.
3. From the expected outcomes and predictions count:
  1. The number of correct predictions for each class.
  2. The number of incorrect predictions for each class, organized by the class that was predicted.

These numbers are then organized into a table, or a matrix as follows:

- **Expected down the side:** Each row of the matrix corresponds to a predicted class.
- **Predicted across the top:** Each column of the matrix corresponds to an actual class.

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)



# Confusion Matrix (Cont.)

- A confusion matrix is an  $N \times N$  matrix, where  $N$  is the number of classes being predicted. For the problem in hand, we have  $N=2$ , and hence we get a  $2 \times 2$  matrix.
- The rows correspond to **the true classes** and the columns correspond to **the predicted classes**.
- Let's work on an example:  
work on an example

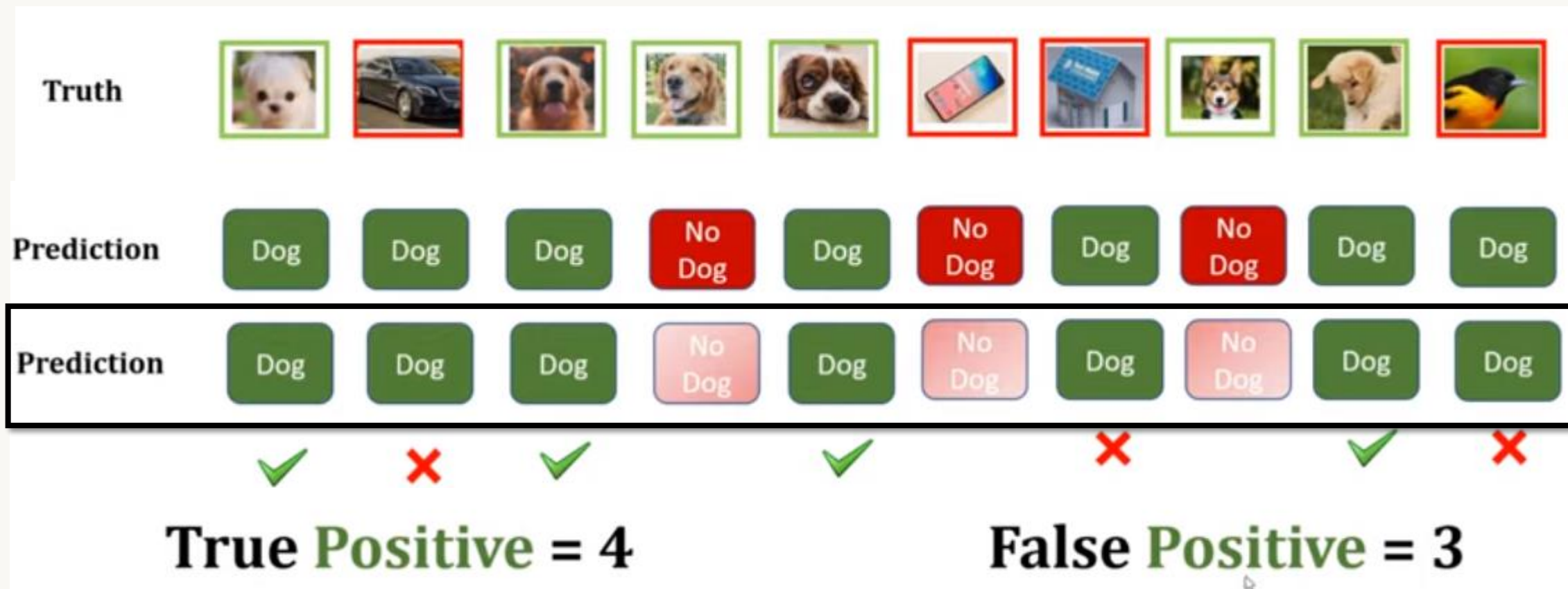
Truth										
Prediction	Dog	Dog	Dog	No Dog	Dog	No Dog	Dog	No Dog	Dog	Dog
	✓	✗	✓	✗	✓	✓	✗	✗	✓	✗

# Confusion Matrix (Cont.)

---

Let's do the Calculations for the **Positive** Class

# Confusion Matrix (Cont.)



# Confusion Matrix (Cont.)

Truth										
Prediction	Dog	Dog	Dog	No Dog	Dog	No Dog	Dog	No Dog	Dog	Dog
	✓	✗	✓	✗	✓	✓	✗	✗	✓	✗

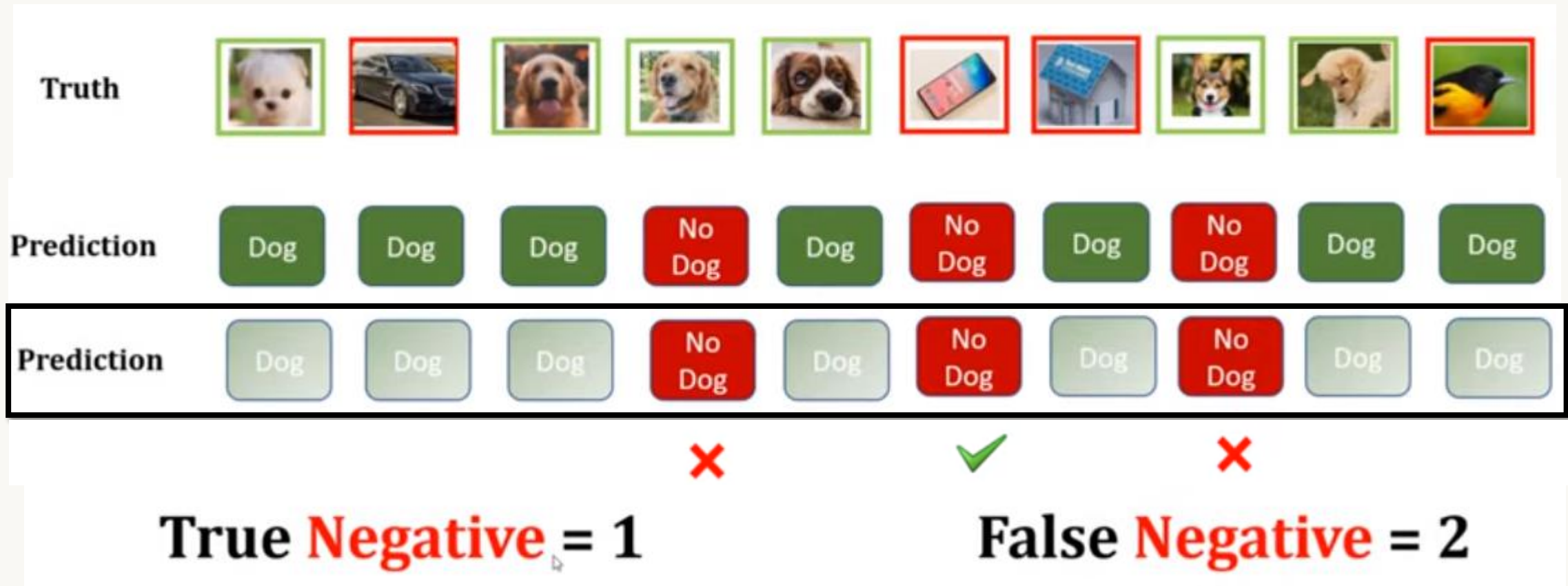
- How many we got right? → 5  
Accuracy →  $5/10 \rightarrow 0.5$
- Precision is: Out of all **dog predictions** how many you got it right? → 4 out of 7 input  
Precision →  $4/7 \rightarrow 0.57$   $\text{Precision} = TP / (TP + FP)$
- Recall is: Out of all **dog truth** how many you got it right? → 4 out of 6 actual  
Recall →  $4/6 \rightarrow 0.67$   $\text{Recall} = TP / (TP + FN)$

# Confusion Matrix (Cont.)

---

Now, let's do the Calculations for the **Negative** Class

# Confusion Matrix (Cont.)



# Confusion Matrix (Cont.)

Truth										
Prediction	Dog	Dog	Dog	No Dog	Dog	No Dog	Dog	No Dog	Dog	Dog
	✓	✗	✓	✗	✓	✓	✗	✗	✓	✗

- How many we got right? → 5  
Accuracy →  $5/10 \rightarrow 0.5$

**Note:** The accuracy is the same because it assesses the number of correct predictions regardless of the class type!!

- Precision is: Out of all **No dog predictions** how many you got it right? → 1 out of 3 input  
Precision →  $1/3 \rightarrow 0.33$   $\text{Precision} = TP / (TP + FP)$
- Recall is: Out of all **No dog truth** how many you got it right? → 1 out of 4 actual  
Recall →  $1/4 \rightarrow 0.25$   $\text{Recall} = TP / (TP + FN)$

# Confusion Matrix (Cont.)

Look at this summarization of Accuracy, Precision, and Recall

		Ground truth		
		+	-	
Predicted	+	True positive (TP)	False positive (FP)	Precision = $TP / (TP + FP)$
	-	False negative (FN)	True negative (TN)	
		Recall = $TP / (TP + FN)$		Accuracy = $(TP + TN) / (TP + FP + TN + FN)$

Now, Let's explore: **What is F1-score?**



# Confusion Matrix: F1-Score

So far, we discussed precision and recall for classification problems and also highlighted the importance of choosing precision/recall basis on our use case.

- What if we are trying to get the best precision and recall at the same time?

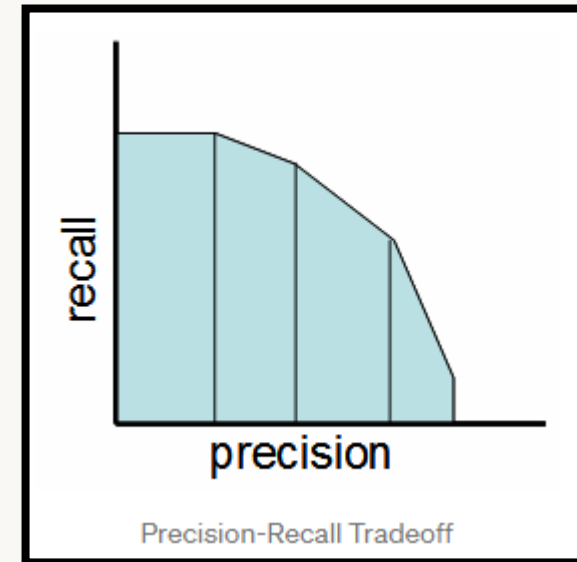
- **F1-Score** (or also referred to as the F-Measure) is the harmonic mean of precision and recall values for a classification problem.
- It is a great measure if you want to find balance between precision and recall for your model and should be used to find a generally good model.
- The formula for F1-Score is as follows:

$$F_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

# Confusion Matrix - F1-Score (Cont.)

---

- Simply stated the **F1 score** sort of maintains a balance between the precision and recall for your classifier.
  - If your precision is low, the F1 is low
  - If the recall is low again your F1 score is low.



# Log Loss

- Log loss (Binary Cross-entropy) is a pretty good evaluation metric for binary classifiers.

*It is used when the output of a classifier is prediction probabilities. **Log Loss** takes into account the uncertainty of your prediction based on how much it varies from the actual label.*

- It is sometimes the optimization objective as well in case of Logistic regression and Neural Networks.
- It is calculated as the negative average of the log of corrected predicted probabilities for each instance.

$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

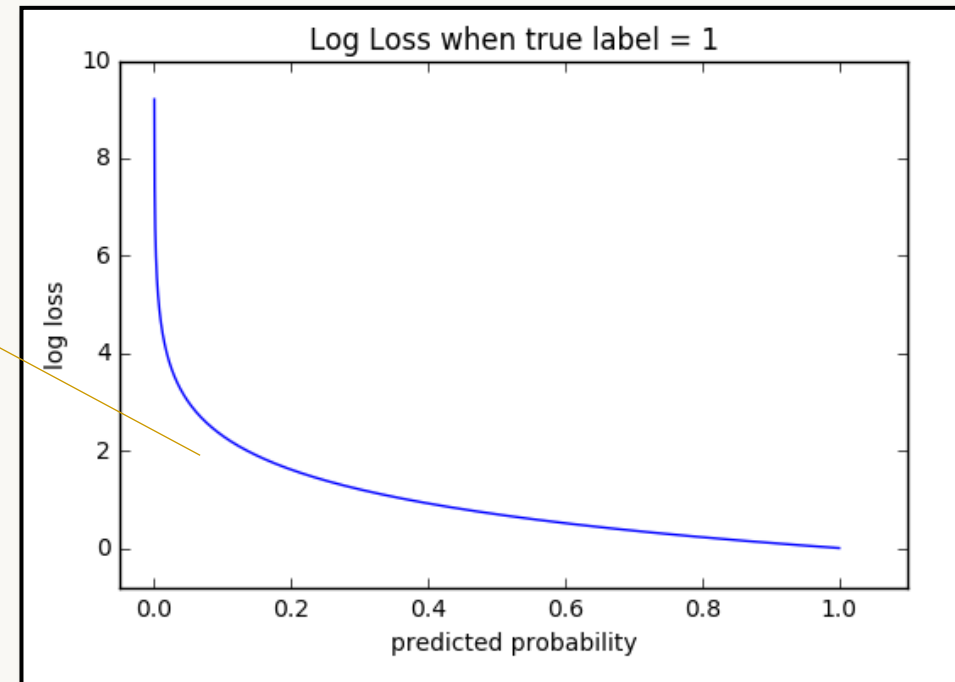
- $p(y_i)$  is predicted probability of positive class
- $1 - p(y_i)$  is predicted probability of negative class
- $y_i = 1$  for positive class and  $y_i = 0$  for negative class (actual values)

# Log Loss (Cont.)

- Binary Log loss for an example is given by the below formula where  $p$  is the probability of predicting **1**.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

As you can see, the log loss decreases as we are fairly certain in our prediction of 1 and the true label is 1.



In general, minimizing Log Loss gives greater accuracy for the classifier.

# Log Loss (Cont.)

---

**Note:**

- When the output of a classifier is **multiclass prediction probabilities**. We generally use **Categorical Cross-entropy** in case of Neural Networks.
- In general, minimizing Categorical cross-entropy gives greater accuracy for the classifier.

# Root Mean Squared Error (RMSE)

---

- **RMSE** is the most popular evaluation metric **used in regression problems**.
- It follows an assumption that error are unbiased and follow a normal distribution.
- It is highly affected by outlier values. Hence, make sure you've removed outliers from your data set prior to using this metric.
- As compared to mean absolute error, RMSE gives higher weightage and punishes large errors.

- RMSE metric is given by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

where, N is Total Number of Observations.

---

Beyond all of the previously mentioned evaluation metrics, there is another method to check the model performance. It is a statistical and robust method that **helps with model selection**.

**Yes! I'm talking about Cross Validation.**

# Cross Validation

---

So, the classical way of:

## Train – validation – test sets

- Not suitable when data is limited
- Alternative is to use (cross validation technique)

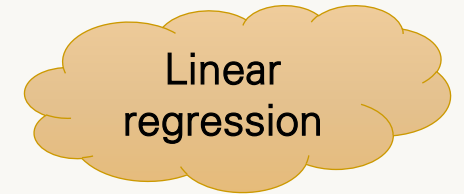
Cross validation isn't a really an evaluation metric which is used openly to communicate model accuracy. But the result of cross validation provides good enough intuitive result to generalize the performance of a model.



# Cross Validation (Cont.)

## ➤ How can we find the best model?

- Split data into [cross validation set], [test set]
- Use cross validation set to train and calculate mean error using one of the folds at a time



<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>	<b>Testing set</b>
---------------	---------------	---------------	---------------	---------------	--------------------

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 2,3,4,5 , measure error using fold 1 , let's say **p1**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,3,4,5 , measure error using fold 2 , let's say **p2**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,4,5 , measure error using fold 3 , let's say **p3**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,5 , measure error using fold 4 , let's say **p4**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,4 measure error using fold 5 , let's say **p5**

Linear regression error = average( $p_1 + p_2 + p_3 + p_4 + p_5$ )

# Cross Validation (Cont.)

## ➤ How can we find the best model?

- Split data into [cross validation set], [test set]
- Use cross validation set to train and calculate mean error using one of the folds at a time

Polynomial  
with degree 2

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>	<b>Testing set</b>
---------------	---------------	---------------	---------------	---------------	--------------------

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 2,3,4,5 , measure error using fold 1 , let's say **p1**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,3,4,5 , measure error using fold 2 , let's say **p2**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,4,5 , measure error using fold 3 , let's say **p3**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,5 , measure error using fold 4 , let's say **p4**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,4 measure error using fold 5 , let's say **p5**

Polynomial with degree 2 error = average( p1+p2+p3+p4+p5)

# Cross Validation (Cont.)

## ➤ How can we find the best model?

- Split data into [cross validation set], [test set]
- Use cross validation set to train and calculate mean error using one of the folds at a time

Polynomial  
with degree 3

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>	<b>Testing set</b>
---------------	---------------	---------------	---------------	---------------	--------------------

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 2,3,4,5 , measure error using fold 1 , let's say **p1**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,3,4,5 , measure error using fold 2 , let's say **p2**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,4,5 , measure error using fold 3 , let's say **p3**

<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,5 , measure error using fold 4 , let's say **p4**













<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>
---------------	---------------	---------------	---------------	---------------

train using folds 1,2,3,4 measure error using fold 5 , let's say **p5**

Polynomial with degree 3 error = average( p1+p2+p3+p4+p5)

# Cross Validation (Cont.)

- How can we find the best model?
  - ♦ select the best algorithm using which has the minimum mean error

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Testing set	
Algorithm	Training error					Cross Validation error	Choice
Linear regression							
Polynomial with degree 2							
Polynomial with degree 3							✓
Polynomial with degree 4							
Polynomial with degree 5							
Polynomial with degree 6							

# Cross Validation (Cont.)

---

- This is called **k-fold cross-validation**, where k is a user-specified number, usually 5 or 10.
- It is the most commonly used version of cross-validation.

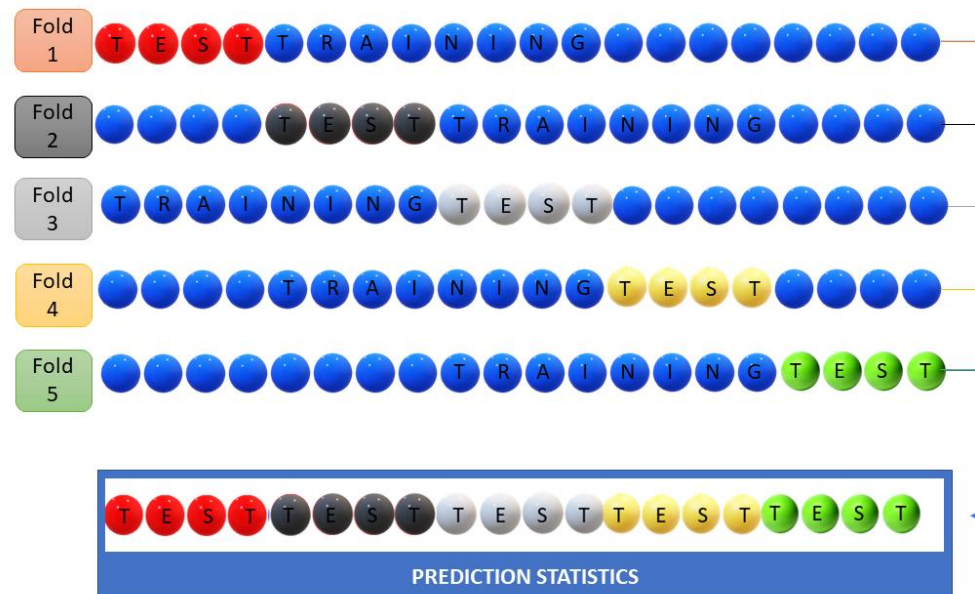
## Note:

- **k-fold cross validation** is widely used to check whether a model is an overfit or not. *If the performance metrics at each of the k times modelling are close to each other and the mean of metric is highest.*
  - Also, **K-fold cross validation** can be **used with any modelling technique**.
- 
- There are other types of cross-validation such as:
    - Leave one out Cross validation (LOOCV)
    - Boot strap cross validation

# Summary

		Ground truth		
		+	-	
Predicted	+	True positive (TP)	False positive (FP)	Precision = $TP / (TP + FP)$
	-	False negative (FN)	True negative (TN)	
		Recall = $TP / (TP + FN)$		Accuracy = $(TP + TN) / (TP + FP + TN + FN)$

## Cross validation



# Appendix

- A place to start and want more experience with Python? Check out Scikit Learn's

evaluation metric functions:

[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)

- More about the Scikit Learn's

Classification Report:

Examples

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
class 0	0.50	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
accuracy			0.60	5
macro avg	0.50	0.56	0.49	5
weighted avg	0.70	0.60	0.61	5

```
>>> y_pred = [1, 1, 0]
>>> y_true = [1, 1, 1]
>>> print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
```

	precision	recall	f1-score	support
1	1.00	0.67	0.80	3
2	0.00	0.00	0.00	0
3	0.00	0.00	0.00	0
micro avg	1.00	0.67	0.80	3
macro avg	0.33	0.22	0.27	3
weighted avg	1.00	0.67	0.80	3

[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html#sklearn.metrics.classification\\_report](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html#sklearn.metrics.classification_report)

---

Next, Regularization

Any questions?