

# Distilled AI

[Back to aman.ai](#)

## Primers • Evaluation Metrics, ROC-Curves and Imbalanced Datasets

- [Introduction](#)
- [Evaluation Metrics for the Classification Problem](#)
  - [Types of Prediction Errors](#)
  - [Accuracy](#)
  - [Confusion Matrix](#)
  - [Precision and Recall](#)
    - [Historical Background](#)
    - [Examples](#)
    - [Applications](#)
    - [Formulae](#)
      - [Memory Map for Precision/Recall Formulae](#)
    - [Precision/Recall Tradeoff](#)
    - [Case Studies](#)
      - [Disease Diagnosis](#)
      - [Search Engine](#)
    - [Precision-Recall Curve](#)
      - [Area Under the PR Curve \(AUPRC\)](#)
      - [Key Takeaways: Precision, Recall and ROC/PR Curves](#)
  - [F<sub>1</sub> Score](#)
    - [Calculating F<sub>1</sub> Score](#)
  - [Sensitivity and Specificity](#)
    - [Precision and Recall vs. Sensitivity and Specificity](#)
  - [Putting It Together: Precision, Recall/Sensitivity, Specificity, and NPV](#)
  - [Calculating Precision, Sensitivity and Specificity](#)
    - [Applications in Information Retrieval](#)
  - [Receiver Operating Characteristic \(ROC\) Curve](#)
    - [Equal Error Rate \(EER\)](#)
    - [Area Under the ROC Curve \(AUROC\)](#)
  - [Detection Error Tradeoff \(DET\) Curve](#)
    - [Comparing ROC and DET Curves](#)
    - [Equal Error Rate \(EER\)](#)
  - [Example Walkthrough with Code](#)
    - [Dataset](#)

- Train a Model for Classification
- Comparing Accuracy vs. Precision-Recall with Imbalanced Data
- ROC Curve vs. Precision-Recall Curve with Imbalanced Data
- Precision-Recall Curve
- Evaluation Metrics for the Regression Problem
  - Mean Absolute Error (MAE)
    - Pros of MAE
    - Cons of MAE
  - Mean Squared Error (MSE)
    - Pros of MSE
    - Cons of MSE
  - Root Mean Squared Error (RMSE)
    - Pros of RMSE
    - Cons of RMSE
  - Root Mean Squared Log Error (RMSLE)
  - R-Squared
  - Adjusted R-Squared
  - Object Detection: IoU, AP, and MAP
    - Intersection Over Union (IoU)
    - Average Precision (AP): Area Under the Curve (AUC)
    - Mean Average Precision (mAP)
- Evaluation Metrics for Generative Text Models
  - Overview
  - Perplexity
  - Burstiness
  - BLEU
    - Example
    - BLEU with Python and `sacreBLEU` Package
  - ROUGE
    - Example
    - ROUGE-L
      - Example
    - ROUGE with Python and `Rouge` Package
  - BLEU vs. ROUGE
    - Goal
    - Summary
      - Similarities
      - Differences
- Evaluation Metrics for Generative Image Models
  - Fréchet Inception Distance
- Evaluation Metrics for Speech Models
- Evaluation Metrics for Clustering
  - Silhouette Score
  - Density-based Clustering Validation (DBCV)

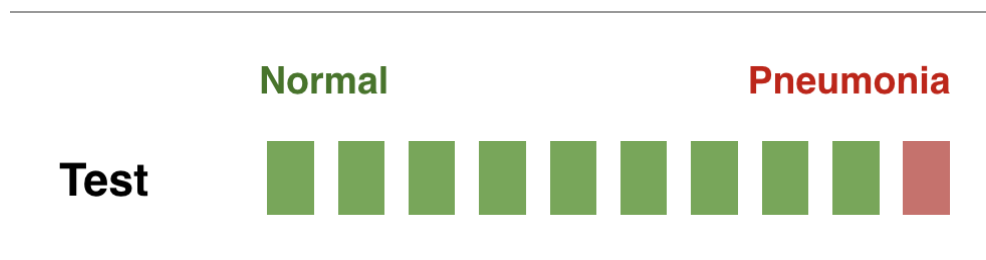
- [Evaluation Metrics for Compression Models](#)
- [Evaluation Metrics for Recommender Systems](#)
  - [Normalized Discounted Cumulative Gain \(NDCG\)](#)
    - [Justification for Using Ranking Quality to Evaluate a Recommendation Engine](#)
    - [Cumulative Gain \(CG\)](#)
    - [Discounted Cumulative Gain \(DCG\)](#)
    - [Normalized Discounted Cumulative Gain \(NDCG\)](#)
    - [Caveats](#)
  - [Summary of Recommender Systems' Evaluation Metrics](#)
- [Evaluation Metrics for GAN-based Models](#)
- [Further Reading](#)
- [References](#)
- [Citation](#)

## Introduction

- Deep learning tasks can be complex and hard to measure: how do we know whether one network is better than another? In some simpler cases such as regression, the loss function used to train a network can be a good measurement of the network's performance.
- However, for many real-world tasks, there are **evaluation metrics** that encapsulate, in a single number, how well a network is doing in terms of real world performance. These evaluation metrics allow us to quickly see the quality of a model, and easily compare different models on the same tasks.
- In this section, we review and compare some of the popular evaluation metrics typically used for classification tasks, and how they should be used depending on the the dataset. Next, we also go over how one can tune the probability thresholds for the particularly metrics. Finally, we'll go through some case studies of different tasks and their metrics.

## Evaluation Metrics for the Classification Problem

- Let's consider a simple binary classification problem, where we are trying to predict if a patient is healthy or has pneumonia. We have a test set with 10 patients, where 9 patients are healthy (shown as green squares) and 1 patient has pneumonia (shown as a red square). The ground truth for the test set is shown below (figure [source](#)):



- We've trained three models for this task (Model1, Model2, Model3), and we'd like to compare the performance of these models. The predictions from each model on the test set are shown below

(figure [source](#)):



## Types of Prediction Errors

- When making a prediction for a two-class classification problem, the following types of errors can be made by a classifier:
  - False Positive (FP)**: predict an event when there was no event. This is also referred to as a **Type I error** in statistical hypothesis testing (defined as the mistaken rejection of an actually true null hypothesis).
  - False Negative (FN)**: predict no event when in fact there was an event. This is also referred to as a **Type II error** in statistical hypothesis testing (defined as the failure to reject a null hypothesis that is actually false).
  - True Positive (TP)**: predict an event when there was an event.
  - True Negative (TN)**: predict no event when in fact there was no event.
- In general, the error type can be interpreted as follows:
  - The **first** word indicates the prediction outcome. If our prediction was **correct**, it's **true**, else it's false.
  - The **second** word indicates the actual prediction. If our prediction was that **an event occurred**, it's **positive**, else it's negative.

## Accuracy

- To compare models, we could first use accuracy, which is the fraction of the number of correctly classified examples relative to the total number of examples:

$$\text{Accuracy} = \frac{\sum_{x_i \in X_{\text{test}}} \mathbb{1}\{f(x_i) = y_i\}}{|X_{\text{test}}|}$$

- For instance, if the classifier is 90% correct, it means that out of 100 instances, it correctly predicts the class for 90 of them.

$$\text{Accuracy} = \frac{\# \text{ of correct predictions}}{\# \text{ of total predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- If we use accuracy as the evaluation metric, it seems that the best model is Model1.

$$\text{Accuracy}(M_1) = \frac{9}{10} \quad \text{Accuracy}(M_2) = \frac{8}{10} \quad \text{Accuracy}(M_3) = \frac{5}{10}$$

- However, accuracy can be misleading if the number of samples per class in the task at hand is unbalanced. Having a dataset with two classes only, where the first class is 90% of the data, and the second completes the remaining 10%. If the classifier predicts every sample as belonging to the first class, the accuracy reported will be of 90% but this classifier is in practice useless. In general, when you have class imbalance (which is **most of the time!**), accuracy is **not** a good metric to use. With imbalanced classes, it's easy to get a high accuracy without actually making useful predictions. So, **accuracy** as an evaluation metrics makes sense only if the class labels are uniformly distributed.

## Confusion Matrix

- Accuracy doesn't discriminate between errors (i.e., it treats misclassifying a patient with pneumonia as healthy the same as misclassifying a healthy patient as having pneumonia). A confusion matrix is a tabular format for showing a more detailed breakdown of a model's correct and incorrect classifications.
- A confusion matrix for binary classification is shown below (figure [source](#)):

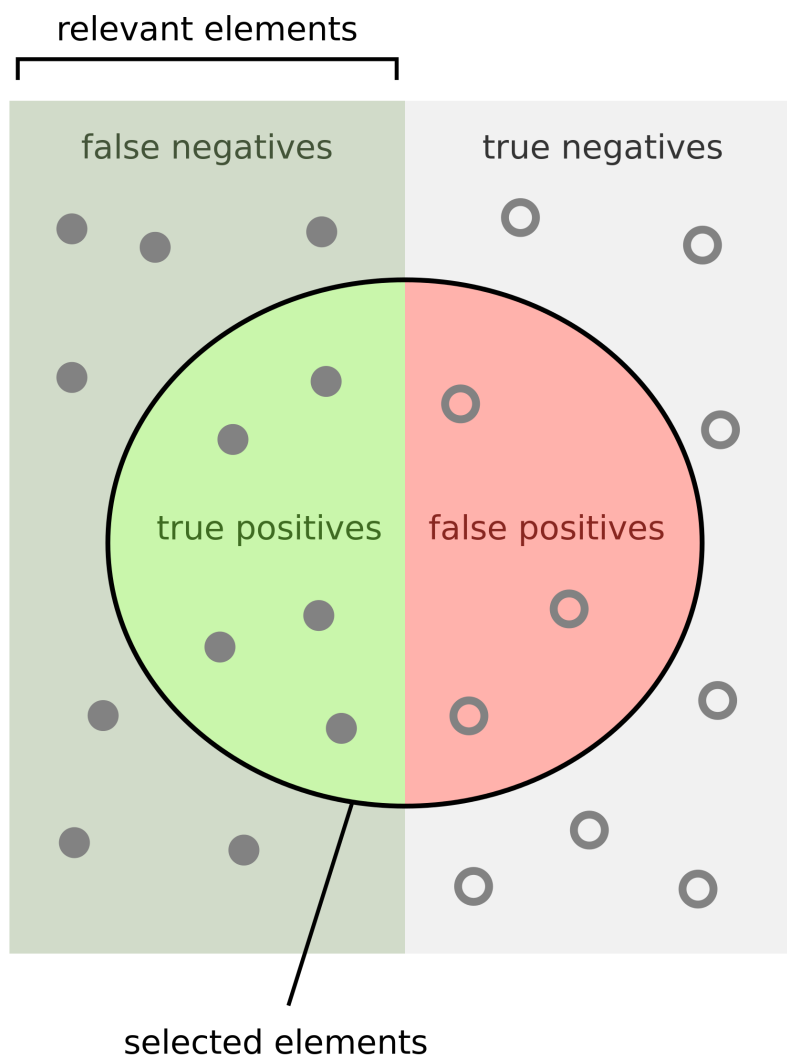
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

# Precision and Recall

- In case of an imbalanced dataset scenario (where you have an abundance of negatives and a dearth of positives), precision and recall are appropriate performance metrics. Both precision and recall are focused on the positive class (the minority class) and are unconcerned with the true negatives (majority class). Put simply, precision and recall are the preferred metrics in case of a class imbalance scenario when you have a lot of negatives and a few positives, for e.g., detecting the wakeword in a typical voice assistant pipeline. In other words, precision and recall make it possible to assess the performance of a classifier on the minority class.
- Precision is defined as the fraction of relevant instances among all retrieved instances.
- Recall, sometimes referred to as **sensitivity**, is the fraction of retrieved instances among all relevant instances.
- Note that precision and recall are computed for each class. They are commonly used to evaluate the performance of classification or information retrieval systems.

*A perfect classifier has precision and recall both equal to 1.*

- It is often possible to calibrate the number of results returned by a model and improve precision at the expense of recall, or vice versa.
- Precision and recall should always be reported together and are not quoted individually. This is because it is easy to vary the sensitivity of a model to improve precision at the expense of recall, or vice versa.
  - As an example, imagine that the manufacturer of a pregnancy test needed to reach a certain level of precision, or of specificity, for FDA approval. The pregnancy test shows one line if it is moderately confident of the pregnancy, and a double line if it is very sure. If the manufacturer decides to only count the double lines as positives, the test will return far fewer positives overall, but the precision will improve, while the recall will go down. This shows why precision and recall should always be reported together.
- The figure below (taken from the Wikipedia article on [precision and recall](#)) shows a graphical representation of precision and recall:



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- Formally, precision and recall can be defined as:
  - Precision:** Out of all the samples marked positive, how many were actually positive (i.e., the true positives)?

- **Recall:** Out of all the samples that are actually positive, how many were marked positive (i.e., the true positives)?
- From the above definitions it is clear that with PR metrics, the focus is on the positive class (also called the relevant class).
- Precision and recall are typically juxtaposed together when reported. Also, it is important to note that Precision and Recall can be interpreted as percentages.
- In the section on [Precision-Recall \(PR\) Curves](#), we explore how to get the best out of these two metrics using PR curves.
- **Key takeaways**
  - Precision: how many **selected** items are **relevant**?
  - Recall: how many **relevant** items are **selected**?

## Historical Background

- This section is optional and offers a historical walk-through of how precision, recall and F1-score came about, so you may skip to the next section if so desired.
- Precision and recall were first defined by the American scientist Allen Kent and his colleagues in their 1955 paper Machine literature searching VIII. Operational criteria for designing information retrieval systems.
- Kent served in the US Army Air Corps in World War II, and was assigned after the war by the US military to a classified project at MIT in mechanized document encoding and search.
- In 1955, Kent and his colleagues Madeline Berry, Fred Luehrs, and J.W. Perry were working on a project in information retrieval using punch cards and reel-to-reel tapes. The team found a need to be able to quantify the performance of an information retrieval system objectively, allowing improvements in a system to be measured consistently, and so they published their definition of precision and recall.
- They described their ideas as a theory underlying the field of information retrieval, just as the second law of thermodynamics “underlies the design of a steam engine, regardless of its type or power rating”.
- Since then, the definitions of precision and recall have remained fundamentally the same, although for search engines the definitions have been modified to take into account certain nuances of human behavior, giving rise to the modified metrics precision @  $k$  and mean average precision (mAP), which are the values normally quoted in information retrieval contexts today.
- In 1979, the Dutch computer science professor Cornelis Joost van Rijsbergen recognized the problems of defining search engine performance in terms of two numbers and decided on a convenient scalar function that combines the two. He called this metric the Effectiveness function and assigned it the letter E. This was later modified to the  $F_1$  score, or  $F_\beta$  score, which is still used today to summarize precision and recall.

## Examples

- Precision and recall can be best explained using examples. Consider the case of evaluating how well does a robot sifts good apples from rotten apples. A robot looks into the basket and picks out all the good apples, leaving the rotten apples behind, but is not perfect and could sometimes mistake a rotten apple for a good apple orange.



- After the robot finishes picking the good apples, precision and recall can be calculated as:
  - **Precision**: number of good apples picked out of all the picked apples.
  - **Recall**: number of good apples picked out of all possible good apples.
- **Precision** is about **exactness**, classifying only one instance correctly yields 100% precision, but a very low recall, it tells us how well the system identifies samples from a given class.
- **Recall** is about **completeness**, classifying all instances as positive yields 100% recall, but a very low precision, it tells how well the system does and identify all the samples from a given class.
- As another example, consider the task of information retrieval. As such, precision and recall can be calculated as:
  - **Precision**: number of relevant documents retrieved out of all retrieved documents.
  - **Recall**: number of relevant documents retrieved out of all relevant documents.

## Applications

- Precision and recall are measured for every possible class in the dataset. So, **precision** and **recall** metrics are relatively much more appropriate (especially compared to accuracy) when dealing with **imbalanced classes**.

*An important point to note is that PR are able to handle class imbalance in scenarios where the positive class (also called the minority class) is rare. If, however, the dataset is imbalanced in such a way that the negative class is the one that's rare, PR curves are sub-optimal and can be misleading. In these cases, [ROC curves](#) might be a better fit.*

- So when do we use PR metrics? Here's the typical use-cases:
  - **When two classes are equally important**: PR would be the metrics to use if the goal of the model is to perform equally well on both classes. Image classification between cats and dogs is a good example because the performance on cats is equally important on dogs.
  - **When minority class is more important**: PR would be the metrics to use if the focus of the model is to identify correctly as many positive samples as possible. Take spam detectors for example, the goal is to find all the possible spam emails. Regular emails are not of interest at all — they overshadow the number of positives.

## Formulae

- Mathematically, precision and recall are defined as,

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

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

- where,

- **TP** is the True Positive Rate, i.e., the number of instances which are relevant and which the model correctly identified as relevant.
- **FP** is the False Positive Rate, i.e., the number of instances which are not relevant but which the model incorrectly identified as relevant.
- **FN** is the false negative rate, i.e., the number of instances which are relevant and which the model incorrectly identified as not relevant.
- The following figure shows a confusion matrix (figure [source](#)), linking the formulae for accuracy, precision, and recall with the components of a confusion matrix.

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

- In the content of the robot sifting good apples from the rotten ones,

$$\text{Precision} = \frac{\# \text{ of picked good apples}}{\# \text{ of picked apples}}$$

$$\text{Recall} = \frac{\# \text{ of picked good apples}}{\# \text{ of good apples}}$$

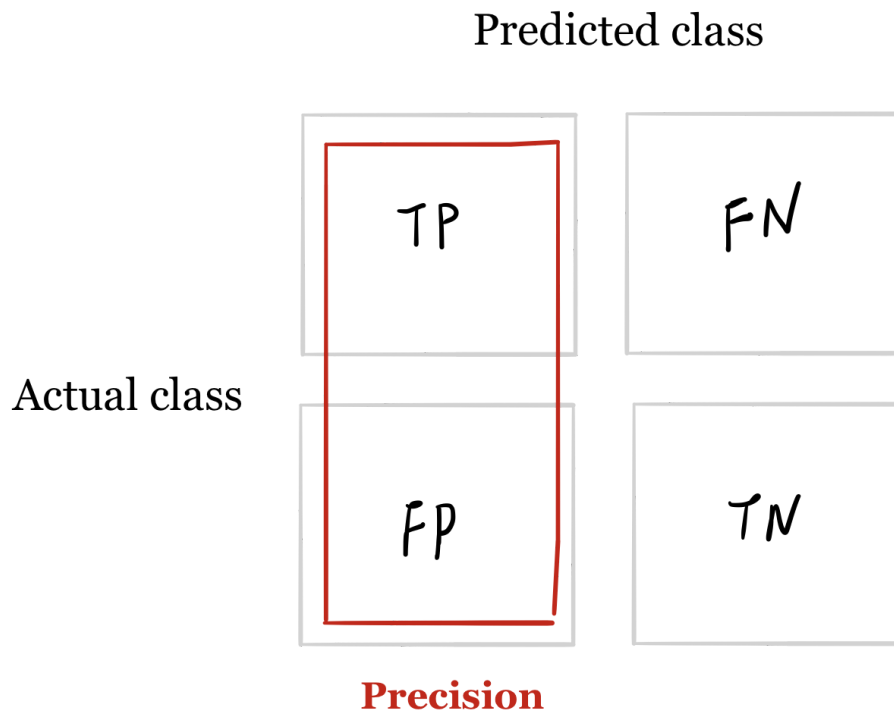
- In the context of information retrieval,

$$\text{Precision} = \frac{\text{retrieved relevant documents}}{\text{all retrieved documents}}$$

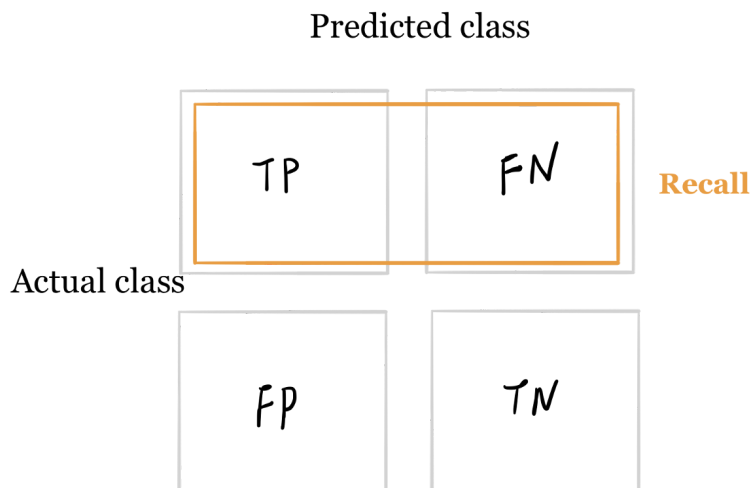
$$\text{Recall} = \frac{\text{retrieved relevant documents}}{\text{all relevant documents}}$$

#### Memory Map for Precision/Recall Formulae

- The formula for precision follows the **triple-P rule**: calculation of Precision related to terms of True Positive and False Positive, which is one half of the confusion matrix.



- The formula for recall is related to the first row of terms in the confusion matrix.

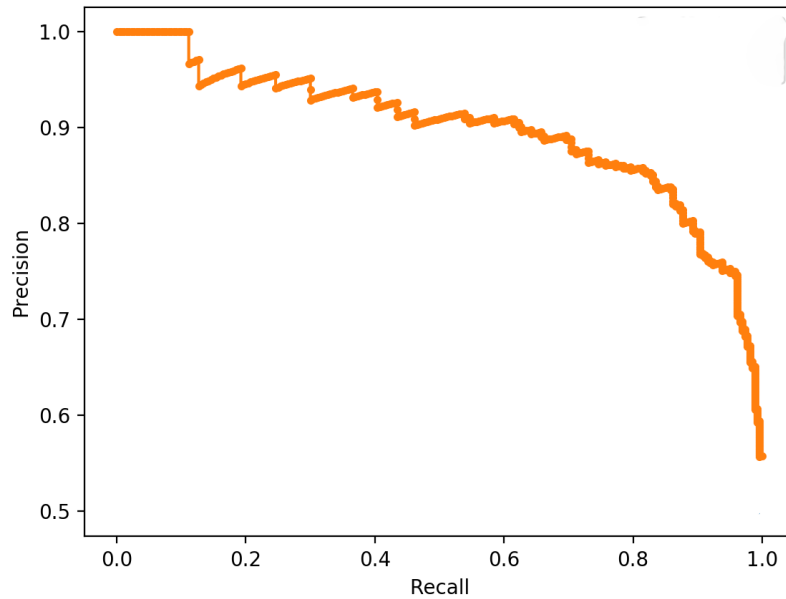


## Precision/Recall Tradeoff

- Because high precision and high recall are what every model optimizes for. However, depending on the problem at hand, you either care about high precision or high recall.
- Examples of high precision:
  - For a model that detects shop lifters, the focus should be on developing a high precision model by reducing false positives (note that precision is given by  $\frac{TP}{TP+FP}$  and since **FP** features in the

denominator, reducing **FP** leads to high precision). This implies that if we tag someone as a shop lifter, we'd like to make sure we do so with high confidence.

- Examples of high recall:
  - In an adult content detection problem, the focus should be on developing a high recall model by reducing false negatives (note that recall is given by  $\frac{TP}{TP+FN}$  and since **FN** features in the denominator, reducing **FN** leads to high recall). This implies that if the model classified a video as good for kids (i.e., not having adult content), it should be marked so with high confidence.
  - In a disease detection scenario, the focus should be on developing a high recall model by reducing false negatives. This implies that if the model classified a patient as not having the disease, it should be done so with high confidence else it can prove fatal.
  - In an autonomous car driving scenario, the focus should be on developing a high recall model by reducing false negatives. This implies that if the model determined that there was no obstacle in the car's surrounding radius, it should be done so with high confidence else fatalities can occur.
- Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. This is called the precision/recall tradeoff. However, in some scenarios, it is important to strike the right balance between both:
  - As an example (from the Wikipedia article on [Precision and Recall](#)), brain surgery provides an illustrative example of the tradeoff. Consider a brain surgeon removing a cancerous tumor from a patient's brain. The surgeon needs to remove all of the tumour cells since any remaining cancer cells will regenerate the tumor. Conversely, the surgeon must not remove healthy brain cells since that would leave the patient with impaired brain function. The surgeon may be more liberal in the area of the brain he removes to ensure he has extracted all the cancer cells. This decision increases recall but reduces precision. On the other hand, the surgeon may be more conservative in the brain he removes to ensure he extracts only cancer cells. This decision increases precision but reduces recall. That is to say, greater recall increases the chances of removing healthy cells (negative outcome) and increases the chances of removing all cancer cells (positive outcome). Greater precision decreases the chances of removing healthy cells (positive outcome) but also decreases the chances of removing all cancer cells (negative outcome).
- In terms of restrictiveness, making the system more restrictive leads to reducing FPs, in turn improving precision. On the other hand, making the system less restrictive leads to reducing FNs, in turn improving recall. Furthermore, in recommender systems, increasing recall has the benefit of showing related results (say, looking up with an item on a restaurant's menu and seeing similar items with it in your search results), leading to improved discovery.
- The following plot ([source](#)) shows the Precision-Recall tradeoff. As we increase the recall rate by adjusting the classification threshold of a model, the precision rate is decreased and vice versa.



## Case Studies

### Disease Diagnosis

- Consider our classification problem of pneumonia detection. It is crucial that we find all the patients that are suffering from pneumonia. Predicting patients with pneumonia as healthy is not acceptable (since the patients will be left untreated).
- Thus, a natural question to ask when evaluating our models is: *Out of all the patients with pneumonia, how many did the model predict as having pneumonia?* The answer to this question is given by the recall.
  - The recall for each model is given by:

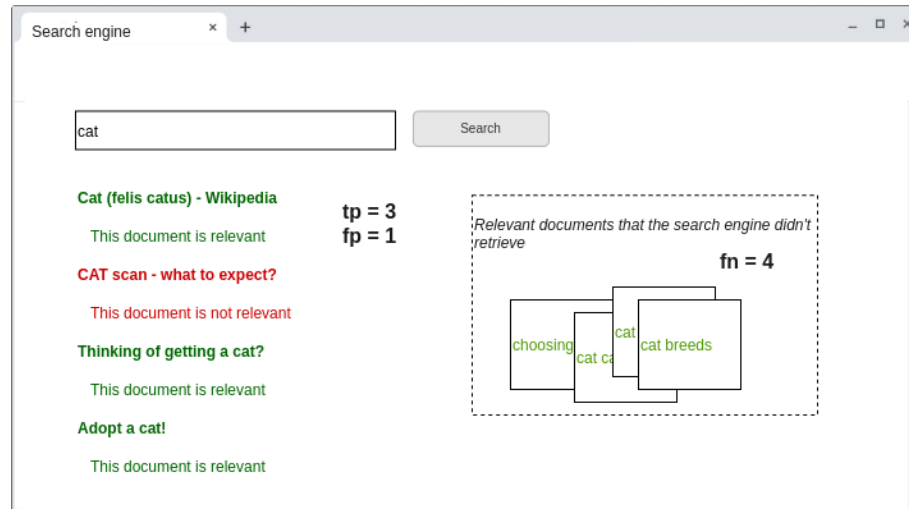
$$\text{Recall}(M_1) = \frac{0}{1} \quad \text{Recall}(M_2) = \frac{1}{1} \quad \text{Recall}(M_3) = \frac{1}{1}$$

- Imagine that the treatment for pneumonia is very costly and therefore you would also like to make sure only patients with pneumonia receive treatment.
- A natural question to ask would be: *Out of all the patients that are predicted to have pneumonia, how many actually have pneumonia?* This metric is the precision.
  - The precision for each model is given by:

$$\text{Precision}(M_1) = \frac{0}{0} \quad \text{Precision}(M_2) = \frac{1}{3} \quad \text{Precision}(M_3) = \frac{1}{6}$$

### Search Engine

- Imagine that you are searching for information about cats on your favorite search engine. You type 'cat' into the search bar.
- The search engine finds four web pages for you. Three pages are about cats, the topic of interest, and one page is about something entirely different, and the search engine gave it to you by mistake. In addition, there are four relevant documents on the internet, which the search engine missed.



- In this case we have three true positives, so  $TP = 3$ . There is one false positive,  $FP = 1$ . And there are four false negatives, so  $FN = 4$ . Note that to calculate precision and recall, we do not need to know the total number of true negatives (the irrelevant documents which were not retrieved).
- The precision is given by,

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP} \\ &= \frac{3}{3 + 1} \\ &= 0.75 \end{aligned}$$

- While the recall is given by,

$$\begin{aligned} \text{Recall} &= \frac{TP}{TP + FN} \\ &= \frac{3}{3 + 4} \\ &= 0.43 \end{aligned}$$

## Precision-Recall Curve

- In the section on [Precision and Recall](#), we see that when a dataset has imbalanced classes, precision and recall are better metrics than accuracy. Similarly, for imbalanced classes, a Precision-Recall curve is more suitable than a [ROC curve](#).

- A Precision-Recall curve is a plot of the **Precision** (y-axis) and the **Recall** (x-axis) for different thresholds, much like the [ROC curve](#). Note that in computing precision and recall there is never a use of the true negatives, these measures only consider correct predictions.

### Area Under the PR Curve (AUPRC)

- Similar to the [AUROC](#), the AUPRC summarizes the curve with a range of threshold values as a single score.
- The score can then be used as a point of comparison between different models on a binary classification problem where a score of 1.0 represents a model with perfect skill.

### Key Takeaways: Precision, Recall and ROC/PR Curves

- **ROC Curve:** summaries the trade-off between the True Positive Rate and False Positive Rate for a predictive model using different probability thresholds.
- **Precision-Recall Curve:** summaries the trade-off between the True Positive Rate and the positive predictive value for a predictive model using different probability thresholds.
- In the same way it is better to rely on **precision** and **recall** rather than **accuracy** in an imbalanced dataset scenario (since it can offer you an incorrect picture of the classifier's performance), a Precision-Recall curve is better to calibrate the probability threshold compared to the ROC curve. In other words, ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets. In both cases, the area under the curve (AUC) can be used as a summary of the model performance.

Metric	Formula	Description
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall/Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1-score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$	Harmonic mean of Precision and Recall

## F<sub>1</sub> Score

- Precision and recall are both useful, but having multiple evaluation metrics makes it difficult to directly compare models. From Andrew Ng's [Machine Learning Yearning](#):

*"Having multiple-number evaluation metrics makes it harder to compare algorithms. Better to combine them to a single evaluation metric. Having a single-number evaluation metric speeds up your ability to make a decision when you are selecting among a large number of classifiers. It gives a clear preference ranking among all of them, and therefore a clear direction for progress."*

- Furthermore, considering the [Precision Recall tradeoff](#), a balancing metric that combines the two terms is helpful. The  $F_1$  score is the harmonic mean of the precision and recall. The  $F_1$  score eases comparison of different systems, and problems with many classes. It is mathematically defined as:

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

- If we consider either precision or recall to be more important than the other, then we can use the  $F_\beta$  score, which is a weighted harmonic mean of precision and recall. This is useful, for example, in the case of a medical test, where a false negative may be extremely costly compared to a false positive. The  $F_\beta$  score formula is more complex:

$$\begin{aligned} F_\beta &= (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}} \\ &= \frac{(1 + \beta^2) \text{TP}}{(1 + \beta^2) \text{TP} + \beta^2 \text{FN} + \text{FP}} \end{aligned}$$

- Since Precision and Recall can be interpreted as percentages (as highlighted in the [Precision and Recall](#) section earlier), their arithmetic mean would be also a percentage. Since  $F_1$  score is actually the harmonic mean of the two; analogously it can be also expressed as a percentage value.
- More on the F-Beta score [here](#).

## Calculating $F_1$ Score

- Imagine that we consider precision and recall to be of equal importance for our purposes. In this case, we will use the  $F_1$ -score to summarize precision and recall together.
- For the above example of [disease diagnosis](#), let's calculate the  $F_1$  score for each model based on the numbers for precision and recall,

$$F_1(M_1) = 0 \quad F_1(M_2) = \frac{1}{2} \quad F_1 = \frac{2}{7}$$

- For the above example of a [search engine](#), let's plug in the numbers for precision and recall into the formula for  $F_1$ -score,



$$\begin{aligned}
 F &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
 &= 2 \times \frac{0.75 \times 0.43}{0.75 + 0.43} \\
 &= 0.55
 \end{aligned}$$

- Note that the  $F_1$  score of 0.55 lies between the precision and recall values of 0.75 and 0.43 respectively. This illustrates how the  $F_1$  score can be a convenient way of averaging the precision and recall in order to condense them into a single number.

## Sensitivity and Specificity

- When we need to express model performance in two numbers, an alternative two-number metric to precision and recall is sensitivity and specificity. This is commonly used for medical stated sensitivity and specificity for a device or testing kit printed on the side of the box, or in the instruction leaflet.
- Sensitivity and specificity can be defined as follows:
  - **Sensitivity:** can be thought of as the extent to which actual positives are not overlooked, so false negatives are few. Note that **sensitivity is the same as recall**.
  - **Specificity:** also called the true negative rate, measures the proportion of actual negatives that are correctly identified as such, i.e., is the extent to which actual negatives are classified as such (so false positives are few).
- Mathematically, sensitivity and specificity can be defined as:

$$\text{Sensitivity} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- In the context of identifying the number of people with a disease,

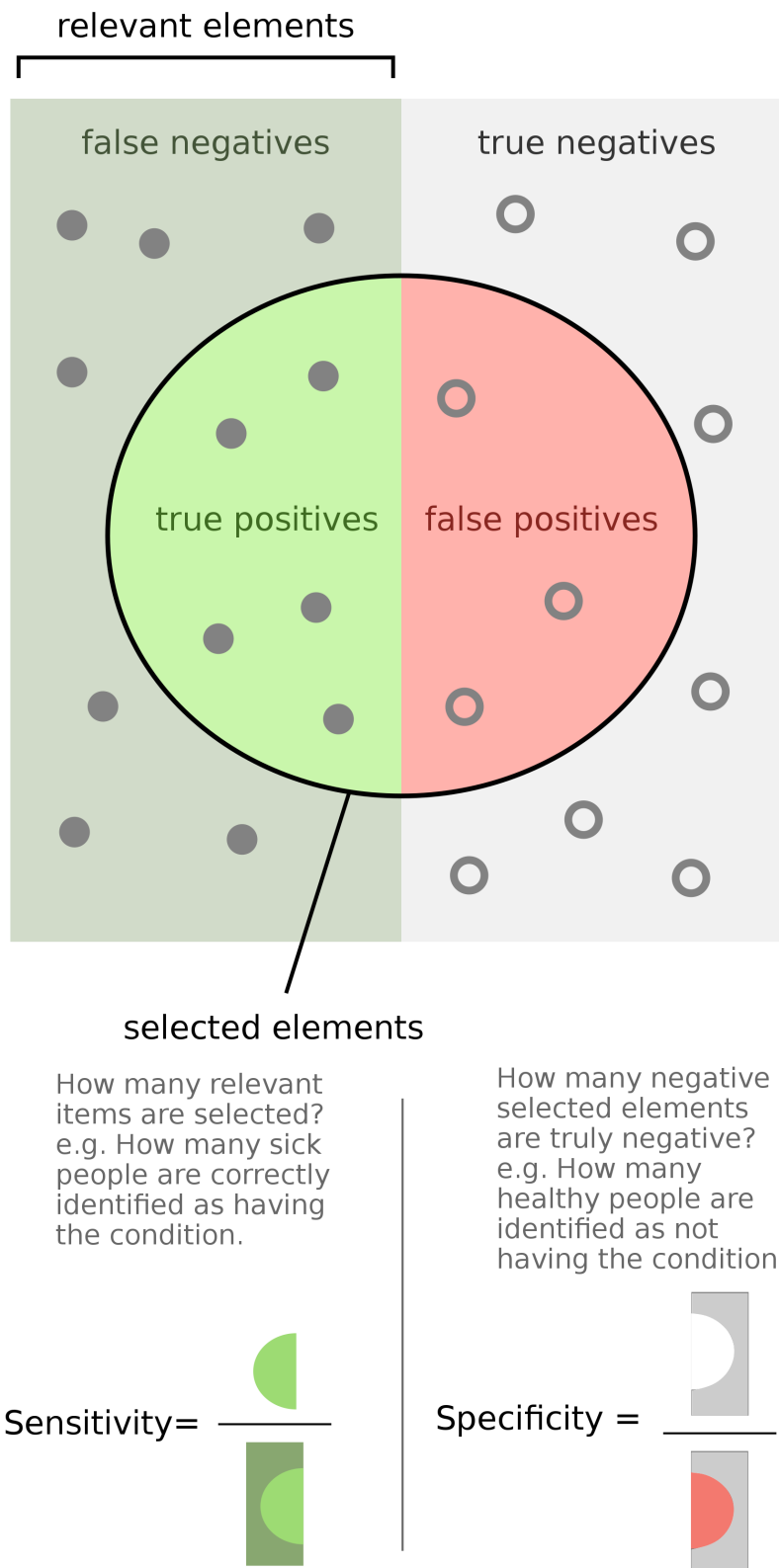
$$\text{Sensitivity} = \frac{\text{true positives}}{\text{number of sick individuals in population}}$$

$$\text{Specificity} = \frac{\text{true negatives}}{\text{number of well individuals in population}}$$

- Sensitivity therefore quantifies the avoiding of false negatives, and specificity does the same for false positives.
- Specificity also uses **TN**, the number of true negatives. This means that sensitivity and specificity use all four numbers in the confusion matrix, as opposed to precision and recall which only use three.
- The number of true negatives corresponds to the number of patients identified by the test as having the disease when they did not have the disease, or alternatively the number of irrelevant documents which the search engine did not retrieve.
- Taking a probabilistic interpretation, we can view specificity as the probability of a negative test given that the patient is well, while the sensitivity is the probability of a positive test given that the patient has

the disease.

- The figure below (taken from the Wikipedia article on [sensitivity and specificity](#)) shows a graphical representation of sensitivity and specificity:



### • Key takeaways

- Sensitivity: how many **relevant** items are **selected**?
- Specificity: how many **negative selected elements** items are truly **negative**?

## Precision and Recall vs. Sensitivity and Specificity

- Sensitivity and specificity are preferred to precision and recall in the medical domain, while precision and recall are the most commonly used metrics for information retrieval. This initially seems strange, since both pairs of metrics are measuring the same thing: the performance of a binary classifier.
- The reason for this discrepancy is that when we are measuring the performance of a search engine, we only care about the returned results, so both precision and recall are measured in terms of the true and false positives. However, if we are testing a medical device, it is important to take into account the number of true negatives, since these represent the large number of patients who do not have the disease and were correctly categorized by the device.
- In medical context, here's scenarios where focusing on one of these two might be important:
  - Sensitivity: the percentage of sick people who are correctly identified as having the condition.
  - Specificity: the percentage of healthy people who are correctly identified as not having the condition.

## Putting It Together: Precision, Recall/Sensitivity, Specificity, and NPV

- The following table ([source](#)) packs all of the metrics that we discussed above:

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) <b>Type II Error</b>	<b>Sensitivity</b> $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) <b>Type I Error</b>	True Negative (TN)	<b>Specificity</b> $\frac{TN}{(TN + FP)}$
		<b>Precision</b> $\frac{TP}{(TP + FP)}$	<b>Negative Predictive Value</b> $\frac{TN}{(TN + FN)}$	<b>Accuracy</b> $\frac{TP + TN}{(TP + TN + FP + FN)}$

## Calculating Precision, Sensitivity and Specificity

- Let's calculate the precision, sensitivity and specificity for the below example of disease diagnosis.
- Suppose we have a medical test which is able to identify patients with a certain disease.
- We test 20 patients and the test identifies 8 of them as having the disease.

- Of the 8 identified by the test, 5 actually had the disease (true positives), while the other 3 did not (false positives).
- We later find out that the test missed 4 additional patients who turned out to really have the disease (false negatives).
- We can represent the 20 patients using the following confusion matrix:

		True state of patient's health	
		Disease	No disease
Test result	Alert	5	3
	No alert	4	8

- The relevant values for calculating precision and recall are  $TP = 5$ ,  $FP = 3$ , and  $FN = 4$ . Plugging in these values into the formulae for precision and recall, we obtain:

$$\begin{aligned}
 \text{Precision} &= \frac{TP}{TP + FP} \\
 &= \frac{5}{5 + 3} \\
 &= 0.625 \\
 \text{Recall} &= \frac{TP}{TP + FN} \\
 &= \frac{5}{5 + 4} \\
 &= 0.56
 \end{aligned}$$

- Next, the relevant values for calculating sensitivity and specificity are  $TP = 5$ ,  $FP = 3$ , and  $TN = 8$ . Note that sensitivity comes out as the same value as recall, as expected:

$$\begin{aligned}
 \text{Sensitivity} &= \frac{TP}{TP + FN} \\
 &= \frac{5}{5 + 4} \\
 &= 0.56
 \end{aligned}$$

- whereas specificity gives:

$$\begin{aligned}
 \text{Specificity} &= \frac{TN}{TN + FP} \\
 &= \frac{8}{8 + 3} \\
 &= 0.73
 \end{aligned}$$

## Applications in Information Retrieval

- Precision and recall are best known for their use in evaluating search engines and other information retrieval systems.

- Search engines must index large numbers of documents, and display a small number of relevant results to a user on demand. It is important for the user experience to ensure that both all relevant results are identified, and that as few as possible irrelevant documents are displayed to the user. For this reason, precision and recall are the natural choice for quantifying the performance of a search engine, with some small modifications.
- Over 90% of users do not look past the first page of results. This means that the results on the second and third pages are not very relevant for evaluating a search engine in practice. For this reason, rather than calculating the standard precision and recall, we often calculate the precision for the first 10 results and call this precision @ 10. This allows us to have a measure of the precision that is more relevant to the user experience, for a user who is unlikely to look past the first page. Generalizing this, the precision for the first  $k$  results is called the precision @  $k$ .
- In fact, search engine overall performance is often expressed as mean average precision, which is the average of precision @  $k$ , for a number of  $k$  values, and for a large set of search queries. This allows an evaluation of the search precision taking into account a variety of different user queries, and the possibility of users remaining on the first results page, vs. scrolling through to the subsequent results pages.

## Receiver Operating Characteristic (ROC) Curve

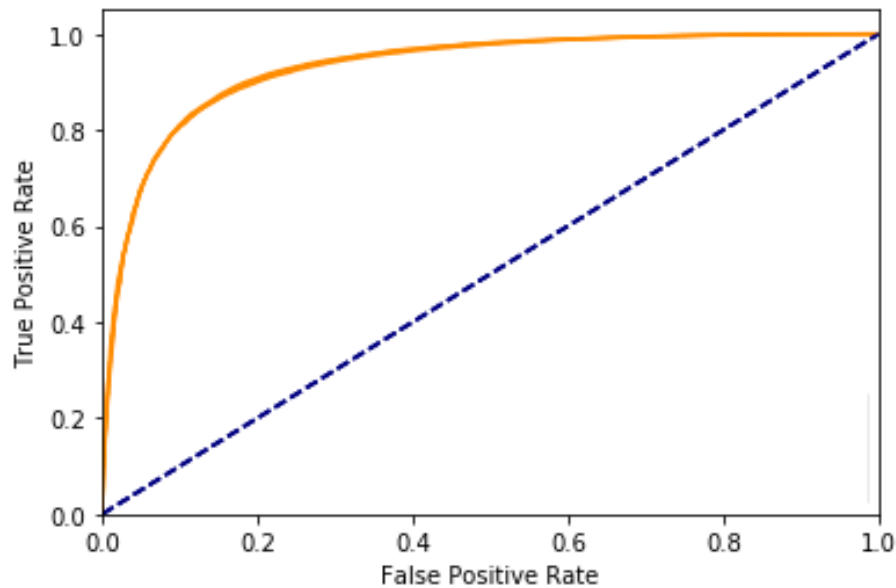
- Suppose we have the probability prediction for each class in a multiclass classification problem, and as the next step, we need to calibrate the threshold on how to interpret the probabilities. Do we predict a positive outcome if the probability prediction is greater than 0.5 or 0.3? The Receiver Operating Characteristic (ROC) curve ROC helps answer this question.
- Adjusting threshold values like this enables us to improve either precision or recall at the expense of the other. For this reason, it is useful to have a clear view of how the False Positive Rate and True Positive Rate vary together.
- The ROC curve shows the variation of the error rates for all values of the manually-defined threshold. The curve is a plot of the **False Positive Rate (also called the False Acceptance Rate) on the X-axis** versus the **True Positive Rate on the Y-axis** for a number of different candidate threshold values between 0.0 and 1.0. A data analyst may plot the ROC curve and choose a threshold that gives a desirable balance between the false positives and false negatives.
  - **False Positive Rate (also called the False Acceptance Rate) on the X-axis:** the False Positive Rate is also referred to as the inverted specificity where specificity is the total number of true negatives divided by the sum of the number of true negatives and false positives.

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{number of negatives}} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- **True Positive Rate on the Y-axis:** the True Positive Rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.

$$\text{True Positive Rate} = \frac{\text{TP}}{\text{number of positives}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Note that both the False Positive Rate and the True Positive Rate are calculated for different probability thresholds.
- As another example, if a search engine assigns a score to all candidate documents that it has retrieved, we can set the search engine to display all documents with a score greater than 10, or 11, or 12. The freedom to set this threshold value generates a smooth curve as below. The figure below ([source](#)) shows a ROC curve for a binary classifier with  $AUC = 0.93$ . The orange line shows the model's false positive and false negative rates, and the dotted blue line is the baseline of a random classifier with zero predictive power, achieving  $AUC = 0.5$ .

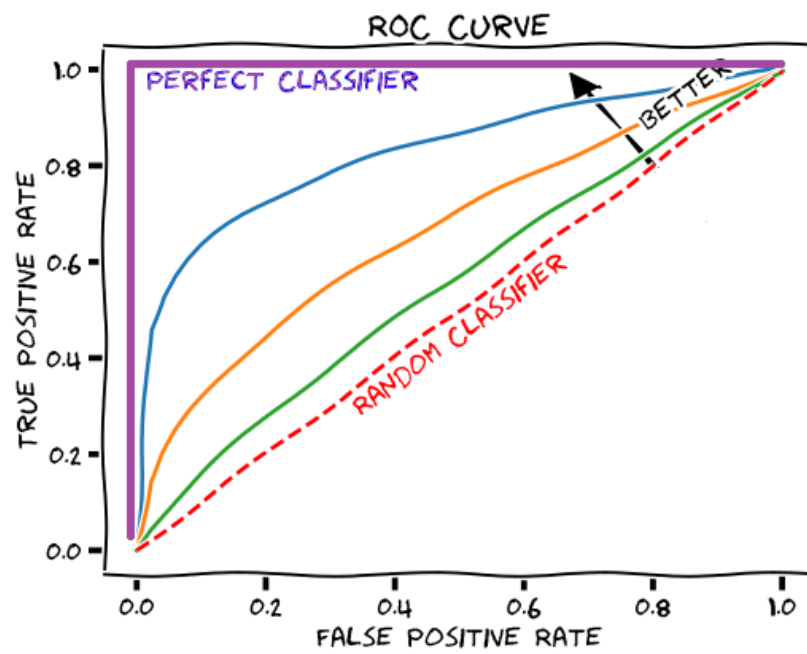


- Note that another way to obtain FPR and TPR is through TNR and FNR respectively, as follows:

$$FPR = 1 - TNR$$

$$TPR = 1 - FNR$$

- The  $y = x$  line in the ROC curve signifies the performance of a random classifier (image [credit](#)). An ROC curve for an ideal/perfect classifier shown in the plot below ([source](#)) would nudge towards the top-left (since higher TPR and lower FPR is desirable) to yield  $AUC \approx 1$ :

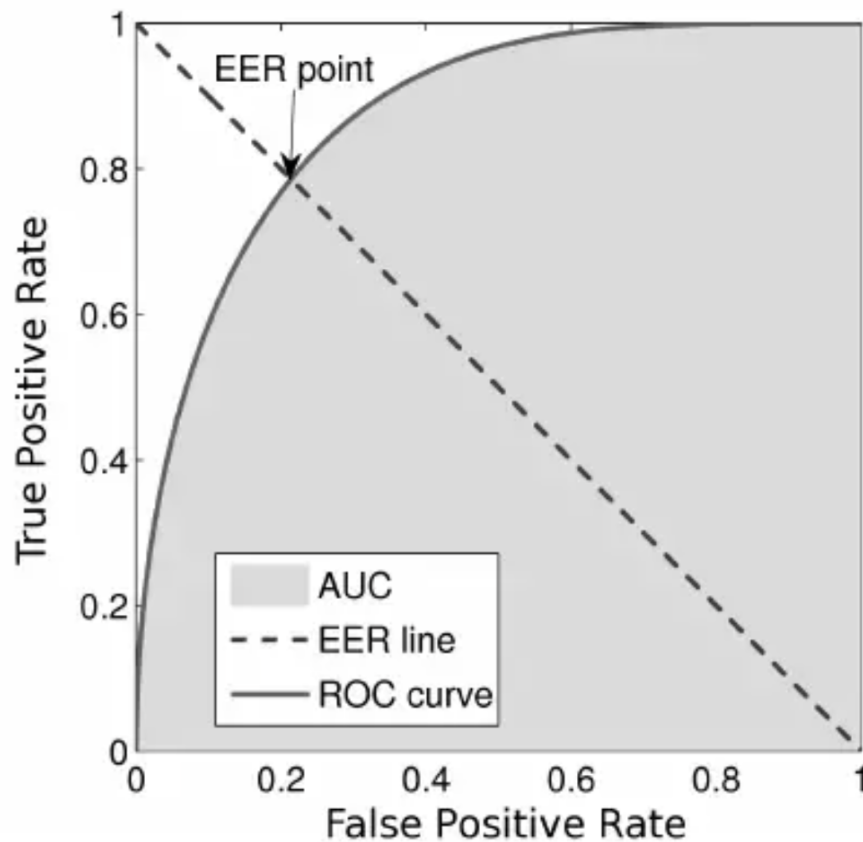




### Equal Error Rate (EER)

- The equal error rate (EER) in ROC curves is the intersection of the  $y + x = 1$  line with the DET curve (figure [source](#)):





### Area Under the ROC Curve (AUROC)

- The area under the ROC curve (AUROC) is a good metric for measuring the classifier's performance. This value is normally between 0.5 (for a bad classifier) and 1.0 (a perfect classifier). The better the classifier, the higher the AUC and the closer the ROC curve will be to the top left corner.

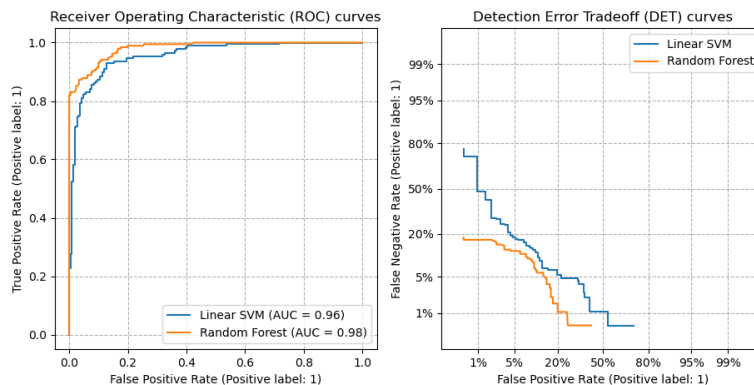
### Detection Error Tradeoff (DET) Curve

- A detection error tradeoff (DET) curve is a graphical plot of error rates for binary classification systems, plotting the false rejection rate (FRR) vs. false acceptance rate (FAR) for different probability thresholds.
- The X- and Y-axes are scaled non-linearly by their standard normal deviates (or just by logarithmic transformation), yielding tradeoff curves that are more linear than ROC curves, and use most of the image area to highlight the differences of importance in the critical operating region.

### Comparing ROC and DET Curves

- Let's compare receiver operating characteristic (ROC) and detection error tradeoff (DET) curves for different classification algorithms for the same classification task.

- DET curves are commonly plotted in normal deviate scale. To achieve this the DET display transforms the error rates as returned by sklearn's `det_curve` and the axis scale using `scipy.stats.norm`.
- The point of this example is to demonstrate two properties of DET curves, namely:
  - It might be easier to visually assess the overall performance of different classification algorithms using DET curves over ROC curves. Due to the linear scale used for plotting ROC curves, different classifiers usually only differ in the top left corner of the graph and appear similar for a large part of the plot. On the other hand, because DET curves represent straight lines in normal deviate scale. As such, they tend to be distinguishable as a whole and the area of interest spans a large part of the plot.
  - DET curves give the user direct feedback of the detection error tradeoff to aid in operating point analysis. The user can deduct directly from the DET-curve plot at which rate false-negative error rate will improve when willing to accept an increase in false-positive error rate (or vice-versa).
- The plots below ([source](#)) example compare the ROC curve on the left with the corresponding DET curve on the right. There is no particular reason why these classifiers have been chosen for the example plot over other classifiers available in scikit-learn.



- To generate DET curves using scikit-learn:

```
import numpy as np
from sklearn.metrics import det_curve

y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])

fpr, fnr, thresholds = det_curve(y_true, y_scores)
# fpr:          array([0.5, 0.5, 0. ])
# fnr:          array([0. , 0.5, 0.5])
# thresholds:   array([0.35, 0.4 , 0.8 ])
```

- Note the formulae to obtain FAR (FPR) and FRR (FNR):

$$\text{FAR} = \text{FPR} = \frac{\text{FP}}{\text{number of negatives}} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{FRR} = \text{FNR} = \frac{\text{FP}}{\text{number of positives}} = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

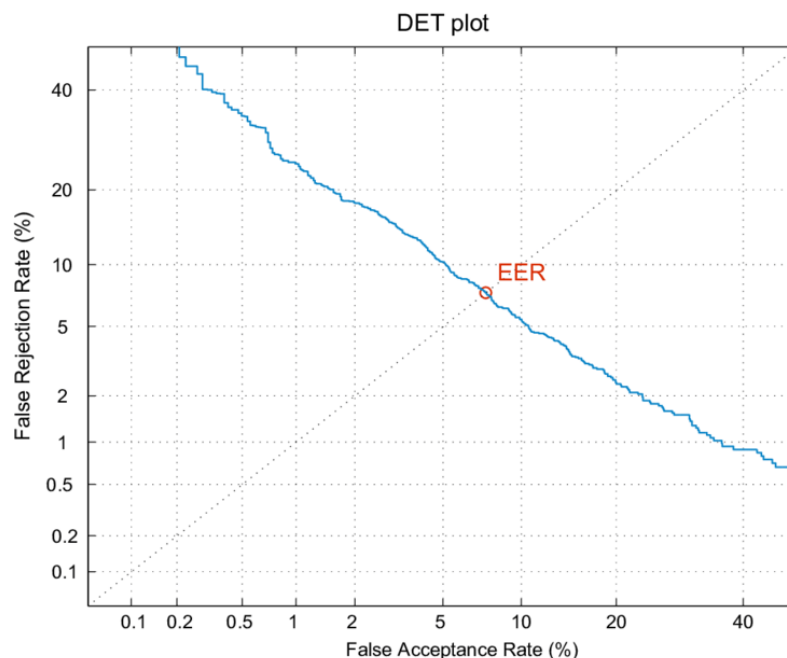
- where, FP: False positive; FN: False Negative; TN: True Negative; TP: True Positive
- Another way to obtain FAR and FRR is through TNR and TPR respectively, as follows:

$$\text{FAR} = 1 - \text{TNR}$$

$$\text{FRR} = 1 - \text{TPR}$$

## Equal Error Rate (EER)

- The equal error rate (EER) in DET curves is the intersection of the  $y = x$  line with the DET curve (figure [source](#)):



## Example Walkthrough with Code

### Dataset

- Let's first generate a 2 class imbalanced dataset

```
X, y = make_classification(n_samples=10000, n_classes=2, weights=[0.95,0.05],
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.2, random_state
```

## Train a Model for Classification

```
model = LogisticRegression()  
model.fit(trainX, trainy)  
predictions = model.predict(testX)
```

## Comparing Accuracy vs. Precision-Recall with Imbalanced Data

```
accuracy = accuracy_score(testy, predictions)  
print('Accuracy: %.3f' % accuracy)
```

- which outputs:

```
Accuracy: 0.957
```

```
print(classification_report(testy, predictions))
```

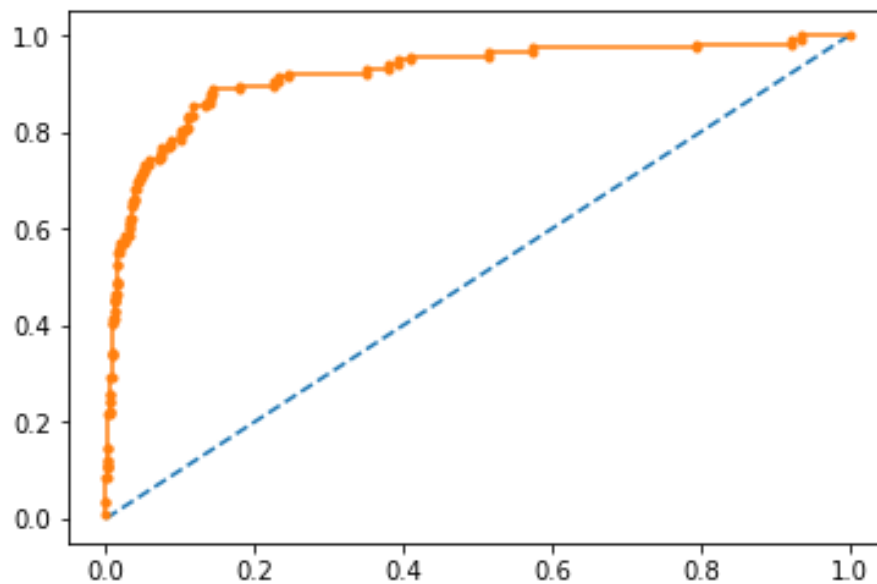
- which outputs:

```
precision recall f1-score support 0 0.96 0.99 0.98 1884 1 0.73 0.41 0.53 116 avg  
/ total 0.95 0.96 0.95 2000
```

## ROC Curve vs. Precision-Recall Curve with Imbalanced Data

```
probs = model.predict_proba(testX)  
probs = probs[:, 1]
```

```
fpr, tpr, thresholds = roc_curve(testy, probs)  
pyplot.plot([0, 1], [0, 1], linestyle='--')  
pyplot.plot(fpr, tpr, marker='.')  
pyplot.show()  
auc_score = roc_auc_score(testy, probs)  
print('AUC: %.3f' % auc_score)
```



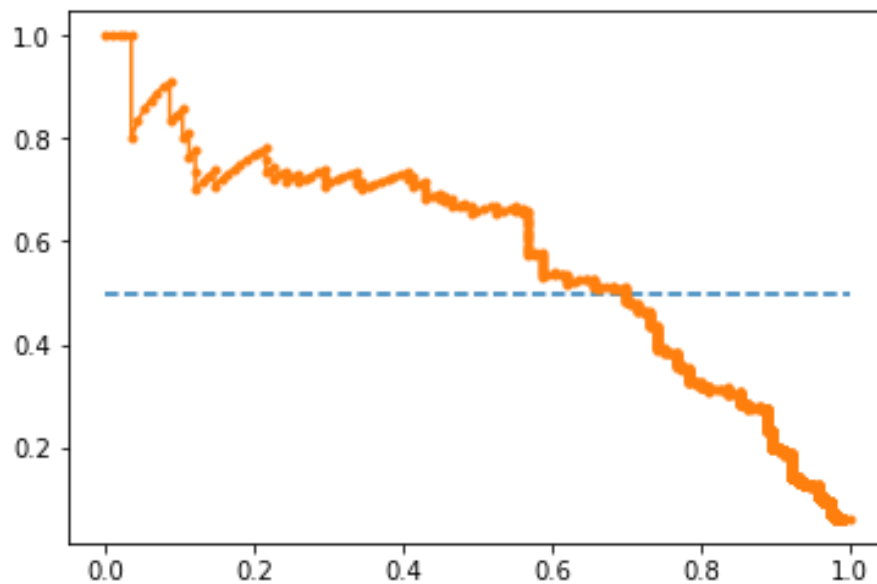
- which outputs:

AUC: 0.920

## Precision-Recall Curve

```
precision, recall, thresholds = precision_recall_curve(testy, probs)
auc_score = auc(recall, precision)
```

```
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
pyplot.plot(recall, precision, marker='.')
pyplot.show()
print('AUC: %.3f' % auc_score)
```



- which outputs:

AUC: 0.577



## Evaluation Metrics for the Regression Problem

- Regression is a type of machine learning problem which helps in finding the relationship between independent and dependent variable.
- Examples include predicting continuous values such as price, rating, etc.

### Mean Absolute Error (MAE)

- MAE is a very simple metric which calculates the absolute difference between actual and predicted values.
- Let's take an example you have input data and output data and use Linear Regression, which draws a best-fit line. To find the MAE for your model, calculate the difference between the actual value and predicted value which yields the absolute error for the current sample. Repeating this for the entire dataset yields the MAE for the model. In other words, sum all the individual errors and divide them by the total number of observations.
- Note that our aim is to minimize MAE because this is a loss function.

$$\text{MAE} = \frac{1}{N} \sum |y - \hat{y}|$$

- where,  $N$  are the total number of data points,  $y$  is the actual output,  $\hat{y}$  is the predicted output and  $|y - \hat{y}|$  is the absolute value of the residual.

```
from sklearn.metrics import mean_absolute_error
print("MAE:", mean_absolute_error(y_test, y_pred))
```

## Pros of MAE

- MAE follows the same units as the output variable so it is easy to interpret.
- MAE is robust to outliers.

## Cons of MAE

- The graph of MAE is not differentiable so we have to apply various optimizers like gradient descent which can be differentiable.

## Mean Squared Error (MSE)

- To overcome the disadvantage of MAE, next metric came as MSE.
- MSE is widely used and differs from MAE just a little bit where MAE utilizes the absolute difference while MSE is based on the squared difference. MSE can thus be obtained by calculating the squared difference between the actual and predicted value.
- Note that MSE calculates the squared distance between the actual and predicted values because it disregards the sign of the error and focuses on only the magnitude. This has the effect of avoiding the cancellation of similar positive and negative terms.

$$\text{MSE} = \frac{1}{n} \sum (y - \hat{y})^2$$

- where,  $(y - \hat{y})^2$  is the square of the difference between the actual and predicted value.
- To obtain RMSE, we can use the NumPy square root function over MSE:

```
from sklearn.metrics import mean_squared_error
print("MSE:", mean_squared_error(y_test, y_pred))
```

## Pros of MSE

- The graph of MSE is differentiable, so you can easily use it as a loss function in deep learning.

## Cons of MSE

- The value you obtain after calculating MSE is a squared unit of the output. For example, if the output units are meters, then the MSE is in meter-squared. This makes interpretation of the loss value difficult.

- The higher the error, more the loss. As such, if you have outliers in the dataset, MSE penalizes the outliers the most since the calculated MSE is larger.

## Root Mean Squared Error (RMSE)

- As the name suggests, RMSE is the square root of MSE. RMSE is probably the most common evaluation metric when working with deep learning techniques.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

```
import numpy as np
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

### Pros of RMSE

- The RMSE uses the same units as the output variable which makes interpretation of loss easy.

### Cons of RMSE

- Not that robust to outliers compared to MAE.

## Root Mean Squared Log Error (RMSLE)

- Taking the log of the RMSE metric slows down the scale of error. The metric is very helpful when you are developing a generative model that is calling the inputs. In that case, the output will vary on a large scale.
- To obtain RMSLE, we can use the NumPy log function over RMSE:

```
print("RMSE", np.log(np.sqrt(mean_squared_error(y_test, y_pred))))
```

## R-Squared

- The R-squared ( $R^2$ ) score is a metric that measures performance by comparing your model with a baseline model.
- $R^2$  calculates the difference between a regression line and a mean line.
- $R^2$  is also known as “coefficient of determination” or sometimes also known as goodness of fit.



$$R^2 = 1 - \frac{SS_r}{SS_m}$$

- where,  $SS_r$  = squared sum error of the regression line;  $SS_m$  is the squared sum error of the mean line.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

- The most common interpretation of  $R^2$  is how well the regression model fits the observed data. For example, an  $R^2$  of 60% reveals that 60% of the data fits the regression model. Generally, a higher  $R^2$  indicates a better fit for the model.
- However, it is not always the case that a high  $R^2$  is good for the regression model. The quality of the statistical measure depends on many factors, such as the nature of the variables employed in the model, the units of measure of the variables, and the applied data transformation. Thus, sometimes, a high  $R^2$  can indicate the problems with the regression model.
- A low  $R^2$  figure is generally a bad sign for predictive models. However, in some cases, a good model may show a small value.

## Adjusted R-Squared

- The disadvantage of the  $R^2$  score is that when adding new features to the data, the  $R^2$  score either increases or remains constant, but it never decreases because it assumes that adding more data causes the variance of data to increase.
- But the problem is when we add an irrelevant feature in the dataset,  $R^2$  sometimes starts increasing, which is incorrect.
- The Adjusted R Squared metric fixes this problem.

$$R_a^2 = 1 - \left[ \left( \frac{n-1}{n-k-1} \right) \times (1 - R^2) \right]$$

- where  $n$  is the number of observations;  $k$  is the number of independent variables;  $R_a^2$  = adjusted  $R^2$ .

```
n=40
k=2
adj_r2_score = 1 - ((1-r2)*(n-1)/(n-k-1))
print(adj_r2_score)
```

- As  $k$  increases by adding some features, the denominator will decrease,  $n - 1$  will remain constant.  $R^2$  score will remain constant or will increase slightly so the complete answer will increase. When we

subtract this from one then the resultant score will decrease – which is what we want when adding irrelevant features to the dataset.

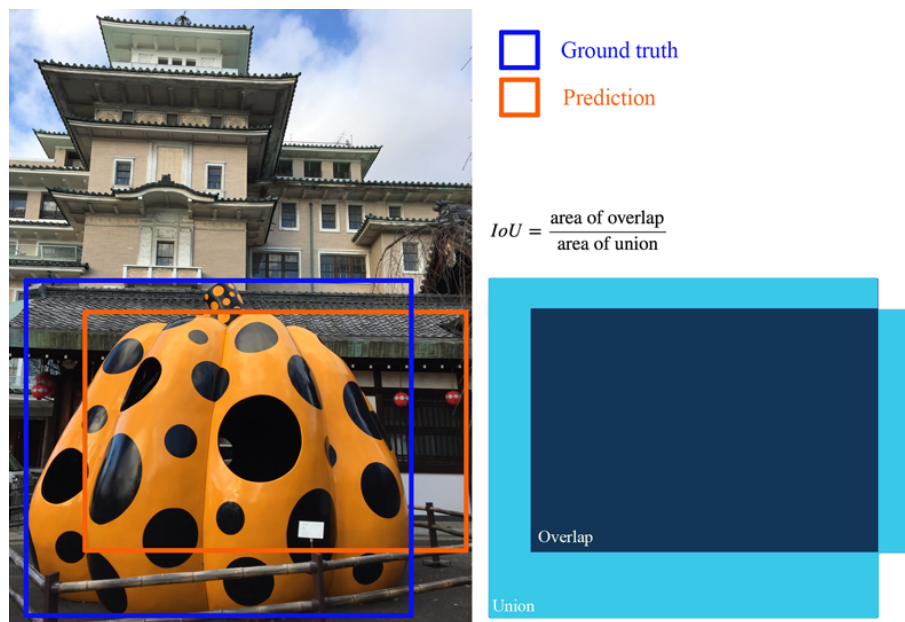
- If we add a relevant feature then the  $R^2$  score will increase and  $(1 - R^2)$  will decrease heavily and the denominator will also decrease so the complete term decreases, and on subtracting from one the score increases.

## Object Detection: IoU, AP, and MAP

- In object detection, two primary metrics are used: intersection over union (IoU) and mean average precision (mAP). Let's walk through a small example.

### Intersection Over Union (IoU)

- Object detection involves finding objects, classifying them, and localizing them by drawing bounding boxes around them. IoU is an intuitive metric that measures the goodness of fit of a bounding box (figure credit to J. Hui's excellent [post](#)):

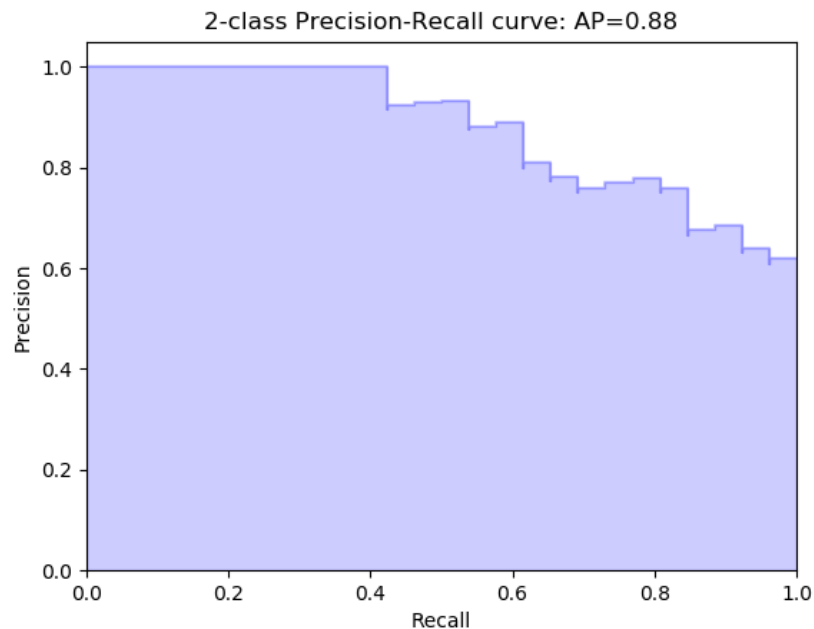


- The higher the IoU, the better the fit. IoU is a great metric since it works well for any size and shape of object. This per-object metric, along with precision and recall, form the basis for the full object detection metric, mean average precision (mAP).

### Average Precision (AP): Area Under the Curve (AUC)

- Object detectors create multiple predictions: each image can have multiple predicted objects, and there are many images to run inference on. Each predicted object has a confidence assigned with it: this is how confident the detector is in its prediction.

- We can choose different confidence thresholds to use, to decide which predictions to accept from the detector. For instance, if we set the threshold to 0.7, then any predictions with confidence greater than 0.7 are accepted, and the low confidence predictions are discarded. Since there are so many different thresholds to choose, how do we summarize the performance of the detector?
- The answer uses a precision-recall curve. At each confidence threshold, we measure the **precision** and **recall** of the detector, giving us one data point. If we connect these points together, one for each threshold, we get a precision recall curve like the following (figure [source](#)):



- The better the model, higher the precision and recall at its points: this pushes the boundary of the curve (the dark line) towards the top and right. We can summarize the performance of the model with one metric, by taking the area under the curve (shown in blue). This gives us a number between 0 and 1, where higher is better. This metric is commonly known as average precision (AP).

## Mean Average Precision (mAP)

- Object detection is a complex task: we want to accurately detect all the objects in an image, draw accurate bounding boxes around each one, and accurately predict each object's class. We can actually encapsulate all of this into one metric: mean average precision (mAP).
- To start, let's compute AP for a single image and class. Imagine our network predicts 10 objects of some class in an image: each prediction is a single bounding box, predicted class, and predicted confidence (how confident the network is in its prediction).
- We start with IoU to decide if each prediction is correct or not. For a ground truth object and nearby prediction, if,
  1. the predicted class matches the actual class, and
  2. the IoU is greater than a threshold,

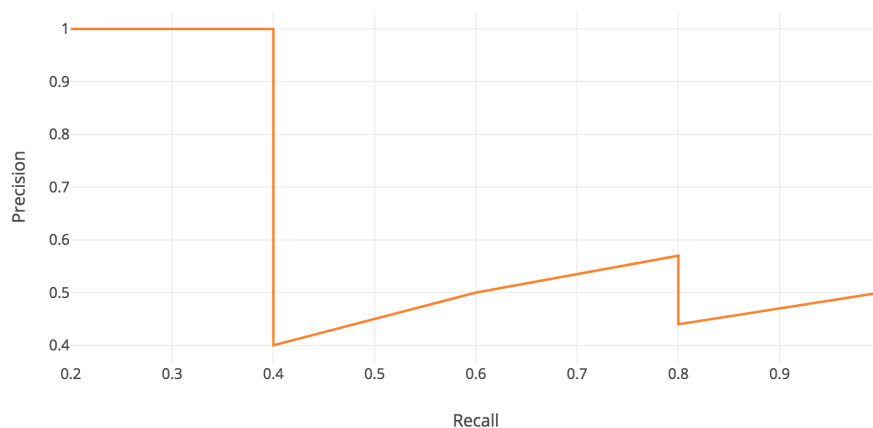
- ... we say that the network got that prediction right (true positive). Otherwise, the prediction is a false positive.
- We can now sort our predictions by their confidence, descending, resulting in the following table. Table of predictions, from most confident to least confident. Cumulative precision and recall shown on the right:

---

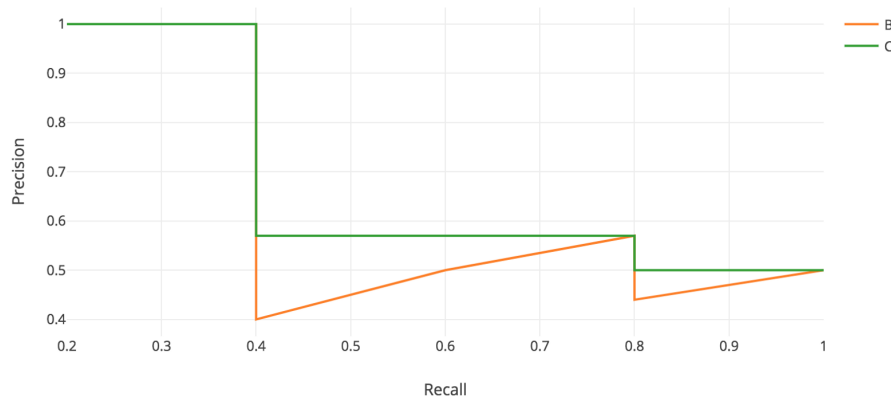
Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

---

- For each confidence level (starting from largest to smallest), we compute the precision and recall up to that point. If we graph this, we get the raw precision-recall curve (figure [source](#)) for this image and class:



- Notice how our precision-recall curve is jagged: this is due to some predictions being correct (increasing recall) and others being incorrect (decreasing precision). We smooth out the kinks in this graph to produce our network's final PR curve for this image and class. The smoothed precision-recall curve (figure [source](#)) used to calculate average precision (area under the curve):



- The average precision (AP) for this image and class is the area under this smoothed curve.
- To compute the mean average precision over the whole dataset, we average the AP for each image and class, giving us one single metric of our network's performance on classification! This is the metric that is used for common object detection benchmarks such as [Pascal VOC](#) and [COCO](#).

## Evaluation Metrics for Generative Text Models

### Overview

- Classification
  - Please refer the section on [Evaluation Metrics for the Classification Problem](#).
- Generative Language Models
  - [Perplexity](#)
  - [Burstiness](#)
- Machine Translation/Captioning
  - [BLEU \(BiLingual Evaluation Understudy\)](#)
  - [CIDEr \(CIDEr: Consensus-based Image Description Evaluation\)](#)
  - [METEOR \(Metric for Evaluation of Translation with Explicit ORdering\)](#)
- Text Summarization
  - [ROUGE \(Recall-Oriented Understudy for Gisting Evaluation\)](#)
- Manual evaluation by humans for text generation (say fluency, grammar, etc.), image generation (say realism based on generated details), recommendation systems (comparative ranking), etc.
  - [Mean Opinion Score \(MOS\)](#)
- NLP Benchmark Suites
  - [GLUE \(General Language Understanding Evaluation\)](#)
  - [SuperGLUE \(Super General Language Understanding Evaluation\)](#)

### Perplexity

- Perplexity (PPL) is the measure of how predictable the text is. It is one of the most common metrics for evaluating the performance of language models. It refers to how well the model is able to predict the next word in a sequence of words. As you'll probably know by now, AI-generated text is procedurally generated; i.e. word-by-word. AI selects the next probable word in a sentence from a K-number of weighted options in the sample.
- Perplexity is based on the concept of entropy, which is the amount of chaos or randomness in a system. So a lower perplexity score indicates that the language model is better at calculating the next word that is likely to occur in a given sequence, while a higher perplexity score indicates that the model is less accurate. Basically, the lower the perplexity, the more predictable it is. This indicates better generalization and performance.
- Wikipedia defines perplexity as: "a measurement of how well a probability distribution or probability model predicts a sample."
- Intuitively, perplexity can be understood as a measure of uncertainty. The perplexity of a language model can be seen as the level of perplexity when predicting the following symbol. Consider a language model with an entropy of three bits, in which each bit encodes two possible outcomes of equal probability. This means that when predicting the next symbol, that language model has to choose among possible options. Thus, we can argue that this language model has a perplexity of 8.
- Mathematically, the perplexity of a language model is defined as:

$$\text{PPL}(P, Q) = 2^{H(P, Q)}$$

- where **H** is the entropy.
- Perplexity can also be defined as the exponent of the negative log-probability. Specifically, perplexity is the exponentiated average negative log-likelihood of a sequence, calculated with exponent base e.
- As a really rough example, let's consider this sentence and try to predict the next token:

*"I picked up the kids and dropped them off at ..."*

- A language model with high perplexity might propose "icicle", "pensive", or "luminous" as answers. Those words don't make sense; it's word salad. Somewhere in the middle might be "the President's birthday party". It's highly unlikely but could be plausible, on rare occasions. But a language model with low perplexity might answer "school" or "the pool". That's an accurate, correct prediction of what likely comes next.
- As you can see, there are varying degrees of plausibility in the output.
- Perplexity is commonly used in NLP tasks such as speech recognition, machine translation, and text generation, where the most predictable option is usually the correct answer.
- For writing generic content that's intended to be standard or ordinary, lower perplexity is the safest bet. Lower the perplexity, the less random the text. Large language models learn to maximize the text probability, which means minimizing the negative log-probability, which in turn means minimizing the perplexity. Lower perplexity is thus desired.
- For more: [Perplexity of fixed-length models](#).

## Burstiness

- Burstiness implies that if a term is used once in a document, then it is likely to be used again. This phenomenon is called burstiness, and it implies that the second and later appearances of a word are less significant than the first appearance.
- Importantly, the burstiness of a word and its semantic content are positively correlated; words that are more informative are also more bursty.
- Burstiness basically measures how predictable a piece of content is by the homogeneity of the length and structure of sentences throughout the text. In some ways, burstiness is to sentences what perplexity is to words.
- Whereas perplexity is the randomness or complexity of the word usage, burstiness is the variance of the sentences: their lengths, structures, and tempos. Real people tend to write in bursts and lulls – we naturally switch things up and write long sentences or short ones; we might get interested in a topic and run on, propelled by our own verbal momentum.
- AI is more robotic: uniform and regular. It has a steady tempo, compared to our creative spontaneity. Humans get carried away and improvise; that's what captures the reader's attention and encourages them to keep reading.
- Burstiness  $b$  is mathematically calculated as:  $b = \left( \frac{\sigma_t/m_t - 1}{\sigma_t/m_t + 1} \right)$  and is bounded within the interval  $[-1, 1]$ . Therefore the hypothesis is  $b_H - b_{AI} \geq 0$ , where  $b_H$  is the mean burstiness of human writers and  $b_{AI}$  is the mean burstiness of AI aka a particular LLM. Corpora with anti-bursty, periodic dispersions of switch points take on burstiness values closer to -1 (usually ones that originate from AI). In contrast, corpora with less predictable patterns of switching take on values closer to 1 (usually ones that originate from humans). Therefore, in the context of AI-generated text detection, the hypothesis is  $b_H - b_{AI} \geq 0$ , where  $b_H$  is the mean burstiness of human writers and  $b_{AI}$  is the mean burstiness of AI aka a particular LLM.

## BLEU

- BLEU, an acronym for **B**ilingual **E**valuation **U**nderstudy was proposed in [BLUE: a Method for Automatic Evaluation of Machine Translation](#) and is predominantly used in machine translation. It quantifies the quality of the machine-generated text by comparing it with a set of reference translations. The crux of the BLEU score calculation is the precision of n-grams (continuous sequence of n words in the given sample of text) in the machine-translated text. However, to prevent the overestimation of precision due to shorter sentences, BLEU includes a brevity penalty factor. Despite its widespread use, it's important to note that BLEU mainly focuses on precision, and lacks a recall component.

*When evaluating machine translation, multiple characteristics are taken into account:*

- *adequacy*
- *fidelity*
- *fluency*

- Mathematically, precision for unigram (single word) is calculated as follows:

$$\text{Precision} = \frac{\text{Number of correct words in machine translation}}{\text{Total words in machine translation}}$$

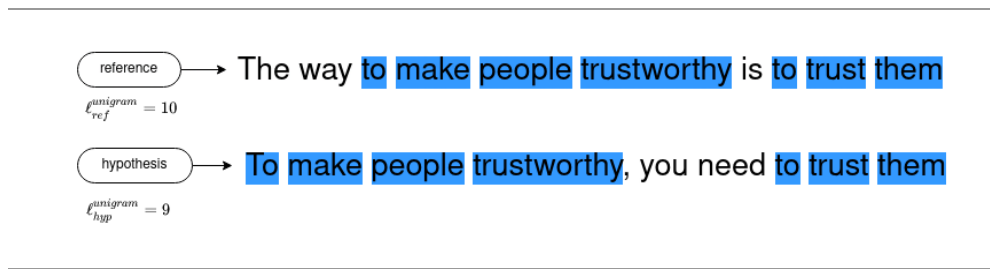
- BLEU extends this idea to consider precision of n-grams. However, BLEU uses a modified precision calculation to avoid the problem of artificially inflated precision scores.
- The equation of BLEU score for n-grams is:

$$\text{BLEU} = \text{BP} * \exp(\sum_{i=1}^n w_i * \log(p_i)) \text{ where,}$$

- **BP** is the brevity penalty (to penalize short sentences).
- $w_i$  are the weights for each gram (usually, we give equal weight).
- $p_i$  is the precision for each i-gram.
- In its simplest form, BLEU is the quotient of the matching words under the total count of words in **hypothesis** sentence (transduction). Referring to the denominator, we can see that BLEU is a precision oriented metric.

$$p_n = \frac{\sum_{n\text{-gram} \in \text{hypothesis}} \text{Count}_{\text{match}}(n\text{-gram})}{\sum_{n\text{-gram} \in \text{hypothesis}} \text{Count}(n\text{-gram})} = \frac{\sum_{n\text{-gram} \in \text{hypothesis}} \text{Count}_{\text{match}}(n\text{-gram})}{\ell_{\text{hyp}}^{n\text{-gram}}}$$

- For example, the matches in the sample sentences are “to”, “make”, “people”, “trustworthy”, “to”, “trust”, “them”



$$p_1 = \frac{7}{9}$$

*Unigram matches tend to measure adequacy while longer n-grams matches account for fluency.*

- As the next step, the calculated precision values for various n-grams are aggregated using a weighted average of the logarithm of precisions.

$$\text{BLEU}_N = \text{BP} \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right)$$

- To counter the disadvantages of precision metric, a brevity penalty is added. Penalty is none, i.e. 1.0, when the hypothesis sentence length is the same as the reference sentence length.
- The brevity penalty **BP** is function of the lengths of reference and hypothesis sentences.

$$\text{BP} = \begin{cases} 1 & \text{if } \ell_{\text{hyp}} > \ell_{\text{ref}} \\ e^{1 - \frac{\ell_{\text{ref}}}{\ell_{\text{hyp}}}} & \text{if } \ell_{\text{hyp}} \leq \ell_{\text{ref}} \end{cases}$$



- BLEU score is a scalar value in range between 0 and 1. A score of 0.6 or 0.7 is considered the best you can achieve. Even two humans would likely come up with different sentence variants for a problem, and would rarely achieve a perfect match.

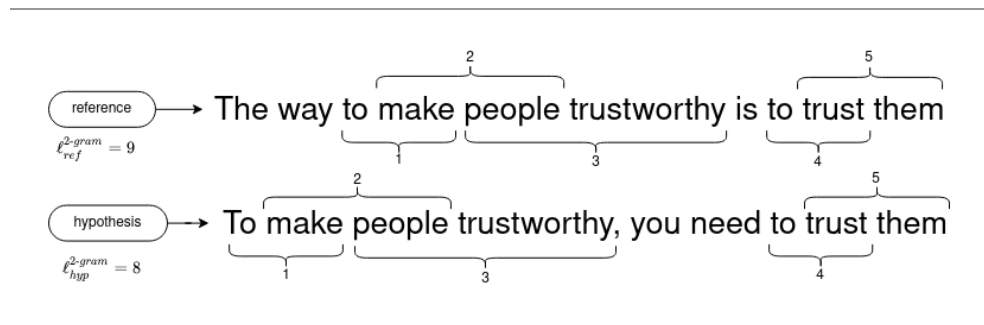
### Example

Type	Sentence	Length
Reference (by human)	The way to make people trustworthy is to trust them.	$\ell_{\text{ref}}^{\text{unigram}} = 10$
Hypothesis/Candidate (by machine)	To make people trustworthy, you need to trust them.	$\ell_{\text{hyp}}^{\text{unigram}} = 9$

- For this example we take parameters as the base line score, described in the paper, with  $N = 4$ , and a uniform distribution, therefore taking  $w_n = \frac{1}{4}$ .

$$\text{BLEU}_{N=4} = \text{BP} \cdot \exp \left( \sum_{n=1}^{N=4} \frac{1}{4} \log p_n \right)$$

- We then calculate the precision  $p_n$  for the different n-grams. For instance, here is an illustration of the bigram (2-gram) matches:



- The following table details the precision values for  $[1, 4]$  n-grams.

n-gram	1-gram	2-gram	3-gram	4-gram
$p_n$	$\frac{7}{9}$	$\frac{5}{8}$	$\frac{3}{7}$	$\frac{1}{6}$

- We then calculate the brevity penalty:

$$\text{BP} = e^{1 - \frac{\ell_{\text{ref}}}{\ell_{\text{hyp}}}} = e^{-\frac{1}{9}}$$

- Finally, we aggregate the precision values across all n-grams, which gives:

$$\text{BLEU}_{N=4} \approx 0.33933$$

## BLEU with Python and `sacreBLEU` Package

- BLEU computation is made easy with the `sacreBLEU` python package.

*For simplicity, the sentences are pre-normalized, removing punctuation and case folding.*

```
from sacrebleu.metrics import BLEU
bleu_scorer = BLEU()

hypothesis = "to make people trustworthy you need to trust them"
reference = "the way to make people trustworthy is to trust them"

score = bleu_scorer.sentence_score(
    hypothesis=hypothesis,
    references=[reference],
)

score.score/100 # sacreBLEU gives the score in percent
```

## ROUGE

- ROUGE score stands for **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation. It was proposed in [ROUGE: A Package for Automatic Evaluation of Summaries](#) and is used primarily for evaluating automatic summarization and, sometimes, machine translation.
- The key feature of ROUGE is its focus on recall, measuring how many of the reference n-grams are found in the system-generated summary. This makes it especially useful for tasks where coverage of key points is important. Among its variants, ROUGE-N computes the overlap of n-grams, ROUGE-L uses the longest common subsequence to account for sentence-level structure similarity, and ROUGE-S includes skip-bigram statistics.
- ROUGE-N specifically refers to the overlap of N-grams between the system and reference summaries.

$$\text{ROUGE} - N = \frac{\text{Number of N-grams in both system and reference summary}}{\text{Total number of N-grams in reference summary}}$$

- ROUGE-L considers sentence level structure similarity naturally and identifies longest co-occurring in-sequence n-grams automatically.
- ROUGE-S includes skip-bigram plus unigram-based co-occurrence statistics. Skip-bigram is any pair of words in their sentence order.

- In its simplest form ROUGE score is the quotient of the matching words under the total count of words in **reference** sentence (summarization). Referring to the denominator, we can see that ROUGE is a recall oriented metric.

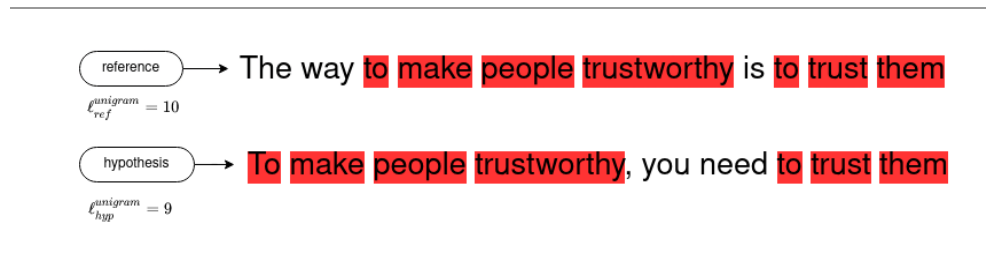
$$\text{ROUGE}_1 = \frac{\sum_{\text{unigram} \in \text{reference}} \text{Count}_{\text{match}}(\text{unigram})}{\sum_{\text{unigram} \in \text{reference}} \text{Count}(\text{unigram})} = \frac{\sum_{\text{unigram} \in \text{reference}} \text{Count}_{\text{match}}(\text{unigram})}{\ell_{\text{ref}}^{\text{unigram}}}$$

## Example

- ROUGE-1 is the ROUGE-N metric applied with unigrams.

Type	Sentence	Length
Reference (by human)	The way to make people trustworthy is to trust them.	$\ell_{\text{ref}}^{\text{unigram}} = 10$
Hypothesis/Candidate (by machine)	To make people trustworthy, you need to trust them.	$\ell_{\text{hyp}}^{\text{unigram}} = 9$

- The following illustrates the computation of ROUGE-1 on the summarization sentences:



$$\text{ROUGE}_1 = \frac{7}{10} = 0.7$$

- Four ROUGE metrics are defined in the original paper: ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S. Here, we focus on the ROUGE-L score.
- ROUGE score is computed as a scalar value in the range  $[0, 1]$ . A ROUGE score close to zero indicates poor similarity between candidate and references. A ROUGE score close to one indicates strong similarity between candidate and references.

## ROUGE-L

- ROUGE-L or  $\text{ROUGE}_{\text{LCS}}$  is based on the length of the longest common subsequence (LCS). To counter the disadvantages of a pure recall metric as in ROUGE-N, ROUGE-L calculates the  $F_\beta$ -score (i.e., weighted harmonic mean with  $\beta$  being the weight), combining the precision score and the recall score.
- The advantages of  $\text{ROUGE}_{\text{LCS}}$  is that it does not simply seek contiguous lexical overlap over n-grams but in-sequence matches (i.e., the overlapping words may not necessarily appear in the same

order). The other (bigger) advantage is that it automatically includes longest in-sequence common n-grams, therefore no predefined n-gram length is necessary.

$$\begin{cases} R_{LCS} &= \frac{LCS(\text{reference}, \text{hypothesis})}{\ell_{\text{ref}}^{\text{unigram}}} \\ P_{LCS} &= \frac{LCS(\text{reference}, \text{hypothesis})}{\ell_{\text{hypothesis}}^{\text{unigram}}} \\ ROUGE_{LCS} &= \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2P_{LCS}} \end{cases}$$

### Example

---

<div style="border: 1px solid black; border-radius: 10px; padding: 2px; display: inline-block; margin-bottom: 5px;">reference</div> $\ell_{\text{ref}}^{\text{unigram}} = 10$	The way to make people trustworthy is to trust them
<div style="border: 1px solid black; border-radius: 10px; padding: 2px; display: inline-block; margin-bottom: 5px;">hypothesis</div> $\ell_{\text{hyp}}^{\text{unigram}} = 9$	<div style="background-color: red; color: black; padding: 2px;">To make people trustworthy, you need to trust them</div>

---

$$\begin{cases} R_{LCS} &= \frac{7}{10} \\ P_{LCS} &= \frac{7}{9} \\ ROUGE_{LCS} &= \frac{(1+\beta^2)49}{70+\beta^2 63} \end{cases}$$

- To give recall and precision equal weights, we take  $\beta = 1$ :

$$ROUGE_{LCS} = \frac{98}{133} \approx 0.73684$$

### ROUGE with Python and `Rouge` Package

- ROUGE computation is made easy with the `Rouge` python package.

*For simplicity, sentences are pre-normalized, removing punctuation and case folding*

```
from rouge import Rouge
rouge_scorer = Rouge()

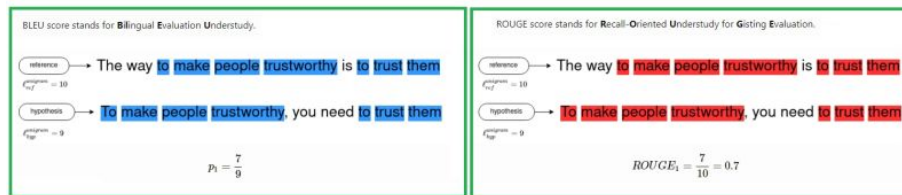
hypothesis = "to make people trustworthy you need to trust them"
reference = "the way to make people trustworthy is to trust them"

score = rouge_scorer.get_scores(
    hyps=hypothesis,
    refs=reference,
```

```
)
score[0] ["rouge-l"] ["f"]
```

## BLEU vs. ROUGE

- BLEU score was first created to automatically evaluate machine translation while ROUGE was created a little later to score the task of auto-summarization.
- Both metrics are calculated using n-gram co-occurrence statistics (i.e., n-gram lexical overlap) and they both range from  $[0, 1]$ , with 0 indicating full dissimilarity and 1 meaning the sentences are exactly the same.
- Despite their relative simplicity, BLEU and ROUGE similarity metrics are quite reliable since they were proven to highly correlate with human judgements.
- The following image ([source](#)) shows a side-by-side comparison of BLEU vs. ROUGE:



## Goal

- Given two sentences, one written by human (reference/gold standard), and a second one generated by a computer (hypothesis/candidate), automatically evaluating the similarity between them is the goal behind these metrics.
- BLEU and ROUGE try to answer this in two different contexts. BLEU for translation between two languages and ROUGE for automatic summarization.
- Here is an example of two similar sentences. We'll use them in the following to illustrate the calculation of both metrics.

Type	Sentence
Reference (by human)	The way to make people trustworthy is to trust them.
Hypothesis/Candidate (by machine)	To make people trustworthy, you need to trust them.

## Summary

### Similarities

- A short summary of the similitudes of the two scoring methods:

- Inexpensive automatic evaluation.
- Count the number of overlapping units such as n-gram, word sequences, and word pairs between hypothesis and references.
- **The more reference sentences the better.**
- Correlates highly with human evaluation.
- Rely on tokenization and word filtering, text normalization.
- Does not cater for different words that have the same meaning — as it measures syntactical matches rather than semantics.

## Differences

BLEU score	ROUGE score
Initially made for translation evaluations ( <b>Bi</b> Lingual <b>E</b> valuation <b>U</b> nderstudy)	Precision oriented score
Initially made for summary evaluations ( <b>R</b> ecall- <b>O</b> riented <b>U</b> nderstudy for <b>G</b> isting <b>E</b> valuation)	Recall oriented score (considering the ROUGE-N version - and not the ROUGE-L version)
One variant (BLUE-N)	Multiple versions (ROUGE-N and ROUGE-L)

## Evaluation Metrics for Generative Image Models

- [Fréchet Inception Distance \(FID\)](#)
- [Inception Score](#)
- [Wasserstein distance](#)

## Fréchet Inception Distance

- This metric compares the statistics of the generated samples and real samples. It models both distributions as multivariate Gaussian. Thus, these two distributions can be compactly represented by their mean  $\mu$  and covariance matrix  $\Sigma$  exclusively. That is:

$$X_r \sim N(\mu_x, \Sigma_x) \text{ and } X_g \sim (\mu_g, \Sigma_g)$$

- These two distributions are estimated with 2048-dimensional activations of the Inception-v3 pool3 layer for real and generated samples respectively.
- Finally, the FID between the real image distribution ( $X_r$ ) and the generated image distribution ( $X_g$ ) is computed as:

$$\text{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$$

- Therefore, lower FID corresponds to more similar real and generated samples as measured by the distance between their activation distributions.

# Evaluation Metrics for Speech Models

- Wakeword Detection (i.e., Keyword Spotting, Trigger Word Detection)/Speaker Recognition
  - [DET Curve: False Accept Rate \(FAR\) vs. False Reject Rate \(FRR\)](#)
- Speech Recognition
  - [Word Error Rate \(WER\)](#)
  - [Character Error Rate \(CER\)](#)
- Text-to-Speech (TTS)
  - Two approaches: (i) subjective evaluation and (ii) objective evaluation.
  - For subjective evaluation the most popular evaluation metric is Mean Opinion Score (MOS), but there are other more complicated tests like [MUSHRA](#).
  - For objective evaluations the most popular test is the simple [Mel Cepstral Distortion \(MCD\)](#) test, but there are more advanced ones. For more details see [Text-to-speech Synthesis by Paul Taylor](#).

# Evaluation Metrics for Clustering

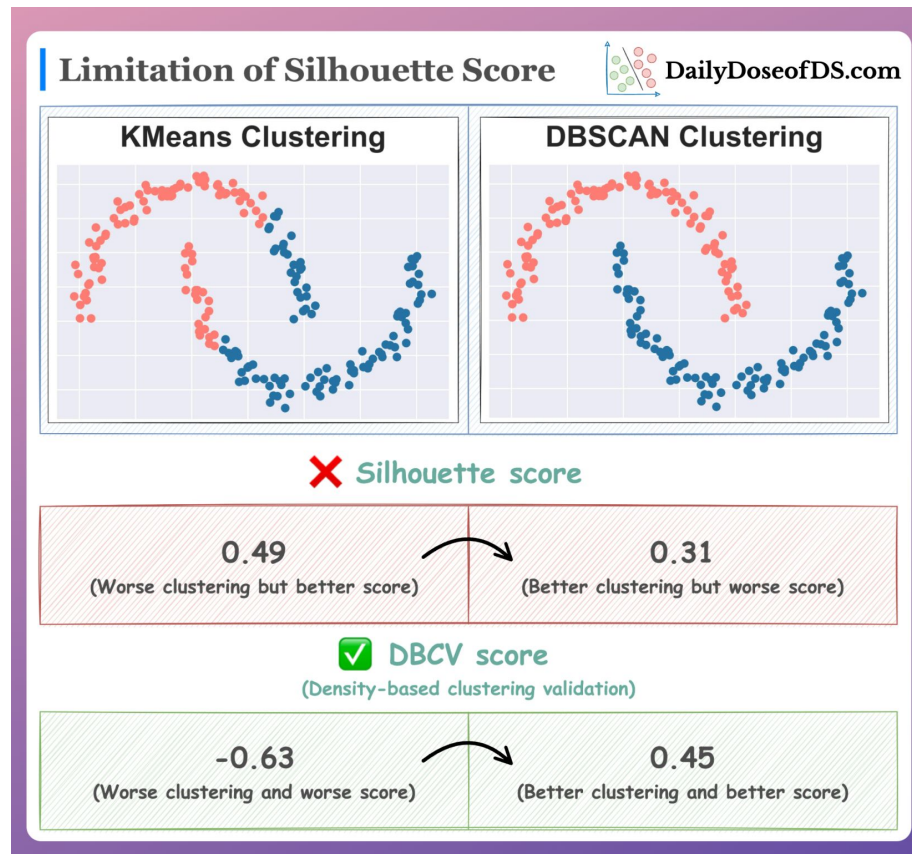
## Silhouette Score

- Silhouette score is a metric to evaluate the performance of clustering algorithm. It uses compactness of individual clusters (intra cluster distance) and separation amongst clusters (inter cluster distance) to measure an overall representative score of how well our clustering algorithm has performed.
- The Silhouette Coefficient is calculated using the mean intra-cluster distance ( $a$ ) and the mean nearest-cluster distance ( $b$ ) for each sample. The Silhouette Coefficient for a sample is  $(b - a) / \max(a, b)$ . To clarify,  $b$  is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is  $2 \leq n\_labels \leq n\_samples - 1$ .
- Scikit-learn offers [sklearn.metrics.silhouette\\_score](#) as an API to calculate the Silhouette score easily.

## Density-based Clustering Validation (DBCV)

- While Silhouette score is commonly used for clustering evaluation, it has a major limitation. The Silhouette score is typically higher for convex (or somewhat spherical) clusters. Thus, using it to evaluate arbitrary-shaped clustering can produce misleading results.
- DBCV is a better metric in such cases.
- Simply put, DBCV computes two values:
  - The density within a cluster.
  - The density between clusters.
- A high density within a cluster and a low density between clusters indicates good clustering results.
- DBCV can also be used when you don't have ground truth labels.
- The following figure ([source](#)) illustrates the effectiveness of DBCV and compares it to the Silhouette score. While the clustering output of KMeans is worse, its Silhouette score is still higher. However,

DBCV produces the expected results.



## Evaluation Metrics for Compression Models

- [Bit Error Rate \(BER\)](#)

## Evaluation Metrics for Recommender Systems

- For an in-depth primer on evaluation metrics for Recommender Systems, please refer our [Recommendation Systems: Evaluation, Metrics, and Loss](#).
- While Mean Average Precision@k (mAP@k) Mean Average Recall@k (mAR@k) are a commonly used evaluation metric, NDCG is a popular measure of ranking quality.

## Normalized Discounted Cumulative Gain (NDCG)

- NDCG is measures the quality of a set of search results. In Information Retrieval, it helps assess document retrieval algorithms.
- NDCG asserts the following:
  1. Very relevant results are more useful than somewhat relevant results which are more useful than irrelevant results (cumulative gain).



2. Relevant results are more useful when they appear earlier in the set of results (discounting).
3. The result of the ranking should be irrelevant to the query performed (normalization).

## Justification for Using Ranking Quality to Evaluate a Recommendation Engine

- A recommendation engine recommends a set of documents from a superset which are most relevant to the user. In that sense, a recommendation engine is simply performing a task of document retrieval. Thus, we can assess a recommendation engine using NDCG.
- To understand NDCG, we need to understand its predecessors: Cumulative Gain (CG) and Discounted Cumulative Gain (DCG).

### Cumulative Gain (CG)

- Every recommendation has a relevance score associated with it. Cumulative Gain is the sum of all the relevance scores in a recommendation set.

$$\text{Cumulative Gain (CG)} = \sum_{i=1}^n \text{relevance}_i$$

- Thus, CG for ordered recommendation set **A** with document relevance scores will be:

$$\begin{aligned} \text{Set A} &= [2, 3, 3, 1, 2] \\ \text{CG}_A &= 2 + 3 + 3 + 1 + 2 = 11 \end{aligned}$$

### Discounted Cumulative Gain (DCG)

- There is a drawback with Cumulative Gain. Consider the following two ordered recommendation sets with relevance scores of individual documents.

$$\begin{aligned} \text{Set A} &= [2, 3, 3, 1, 2] \\ \text{Set B} &= [3, 3, 2, 2, 1] \\ \text{CG}_A &= 2 + 3 + 3 + 1 + 2 = 11 \\ \text{CG}_B &= 3 + 3 + 2 + 2 + 1 = 11 \end{aligned}$$

- We know that Set **B** is better than Set **A** as it is recommending in decreasing order of relevance, but as per the Cumulative Gain, both sets are equally good. What exactly lacking is the use of position along with relevance scores. DCG fills this gap. The computation involves discounting the relevance score by dividing it with the log of the corresponding position.

$$\text{DCG} = \sum_{i=1}^n \frac{\text{relevance}_i}{\log_2(i + 1)}$$

- Alternatively, it can also be computed using the below expression.

$$DCG = \sum_{i=1}^n \frac{2^{\text{relevance}_i} - 1}{\log_2(i + 1)}$$

- This second expression penalizes heavily as compared to the first one if the document with higher relevance is ranked lower. Depending on the application you can choose either one of the expressions to compute the DCG and NDCG.

*If the relevance scores are binary, i.e. either 0 or 1, DCG and NDCG yield the same result.*

- Let us compute the DCG for both ordered sets using the first expression.

$$DCG_A = \frac{2}{\log_2(1 + 1)} + \frac{3}{\log_2(2 + 1)} + \frac{3}{\log_2(3 + 1)} + \frac{1}{\log_2(4 + 1)} + \frac{2}{\log_2(5 + 1)} \approx 6.64$$

$$DCG_B = \frac{3}{\log_2(1 + 1)} + \frac{3}{\log_2(2 + 1)} + \frac{2}{\log_2(3 + 1)} + \frac{2}{\log_2(4 + 1)} + \frac{1}{\log_2(5 + 1)} \approx 7.14$$

$$DCG_A < DCG_B$$

- The DCG results for the above example is aligned with our intuition. Set **B** is better than Set **A**.

## Normalized Discounted Cumulative Gain (NDCG)

- DCG seems a good measure at first as it takes position significance into account. However, it is still not complete. Depending on various factors, the number of recommendations served may vary for every user. Thus, the DCG will vary accordingly. We need a score which has a proper upper and lower bounds so that we can take a mean across all the recommendations score to report a final score. NDCG brings in this normalization.
- For every recommendation set, to compute NDCG, we need to compute:
  1. DCG of the recommended order
  2. DCG of the ideal order (iDCG).
- NDCG is then the ratio of DCG of recommended order to DCG of ideal order.

$$NDCG = \frac{DCG}{iDCG}$$

- This ratio will always be in the range  $[0, 1]$ .
- Consider the following ordered recommendation set served to one of the users.

**Recommendations Order** = [2, 3, 3, 1, 2]

- The ideal order for this recommendation set will be:

**Ideal Order** = [3, 3, 2, 2, 1]

- The corresponding DCG scores using the first expression:

$$\text{DCG} = \frac{2}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{3}{\log_2(3+1)} + \frac{1}{\log_2(4+1)} + \frac{2}{\log_2(5+1)} \approx 6.64$$

$$\text{iDCG} = \frac{3}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{2}{\log_2(3+1)} + \frac{2}{\log_2(4+1)} + \frac{1}{\log_2(5+1)} \approx 7.14$$

- Thus, the NDCG for this recommendation set will be:

$$\text{NDCG} = \frac{\text{DCG}}{\text{iDCG}} = \frac{6.64}{7.14} \approx 0.93$$

*To evaluate a recommendation engine, compute a mean of NDCG for the recommendations served to the test set of the users.*

## Caveats

- NDCG does not penalize irrelevant search results. In our case, it didn't penalize  $D_4$ , which had zero relevance according to the human rater.
- Another result set may not include a particular document as part of the recommendation set (say, due to varying length results), but it would still have the same NDCG score. As a remedy, the human rater could assign a negative relevance score to that document.

## Summary of Recommender Systems' Evaluation Metrics

- In [A survey on news recommender systems](#) (2020), Raza and Ding offer a summary of the definitions for a range of evaluation metrics in the context of recommender systems:

=		
Metric	Description	Type
Precision	The proportion of relevant recommended items over total recommended items.	Accuracy
Recall	The proportion of relevant recommended items over total relevant items.	Accuracy
F1-score	Weighted average of the precision and recall.	Accuracy
Customer Satisfaction Index	The satisfaction degree of a user on the recommendations (Xia et al. 2010).	Beyond-accuracy
Mean Reciprocal Rank (MRR)	The multiplicative inverse of the rank of the first correct item.	Ranking accuracy
Mean Average Precision (MAP)	The average precisions across all relevant queries.	Ranking accuracy
$\overline{\text{Rank}}$	The percentile-ranking of article within the ordered list of all articles.	Ranking accuracy
Cumulative rating	The total relevance of all documents at or above each rank position in the top k.	Ranking accuracy
Success @ k	A current news item that is in sequence and in a set of recommended news items.	Ranking accuracy

Metric	Description	Type
Personalized @ k	A current news item that is in a given sequence and in a set of recommended news items without popular items ( <a href="#">Garcin et al. 2013</a> ).	Personalization accuracy
Novelty @ k	The ratio of unseen and recommended items over the recommended items.	Novelty, beyond-accuracy
Diversity	The degree of how much dissimilar recommended items are for a user.	Diversity, beyond-accuracy
Binary Hit rate	The number of hits in an n-sized list of ranked items over the number of users for whom the recommendations are produced.	Ranking accuracy
Log-loss	To measure the performance of a classification model where the prediction input is a probability value between 0 and 1.	Accuracy
Average Reciprocal Hit-rate	Each hit is inversely weighted relative to its position in top-N recommendations.	Ranking accuracy
Root-mean-square error (RMSE)	Difference between the predicted and the actual rating.	Accuracy
Click-through rate (CTR)	The likelihood of a news item that will be clicked.	Accuracy
Discounted Cumulative Gain (DCG)	The gain of an item according to its position in the result list of a recommender.	Ranking accuracy
Area under curve (AUC)	A ROC curve plots recall (True Positive Rate) against fallout (False Positive Rate).	Accuracy
Saliency	To evaluate if a news entity is relevant for a text document ( <a href="#">Cucchiarelli et al. 2018</a> ).	Beyond-accuracy
Future-Impact	To evaluate how much user attention (views or shares) each news story may receive in the future and is measured between recency and relevancy ( <a href="#">Chakraborty et al. 2019</a> ).	Beyond-accuracy

## Evaluation Metrics for GAN-based Models

- Adjusted Rand Index (ARI) is a widely used metric for evaluating clustering performance. More on this in [sklearn.metrics.adjusted\\_rand\\_score](#).

## Further Reading

Here are some (optional) links you may find interesting for further reading:

- [An empirical study on evaluation metrics of generative adversarial networks \(paper\)](#)
- [How to measure GAN performance? \(blog post\)](#)
- [Are GANs Created Equal? A Large-Scale Study \(paper\)](#)
- [Pros and Cons of GAN Evaluation Measures \(paper\)](#)
- [How to Evaluate GANs \(blog post\)](#)
- [Evaluation Metrics for Language Modeling \(article\)](#)

- [Evaluation Metrics for Language Models \(paper\)](#)
- [Analytics Vidhya: Know The Best Evaluation Metrics for Your Regression Model](#)
- [The Ultimate Performance Metric in NLP](#)

## References

- [Speech and Language Processing \(2019\)](#) by Jurafsky and Martin.
- [A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation \(2005\)](#) by Goutte and Gaussier.
- [Information Retrieval \(2nd ed.\) \(1979\)](#) by Van Rijsbergen.
- [Machine literature searching VIII. Operational criteria for designing information retrieval systems \(1955\)](#) by Kent et al.
- [CS229 Machine Learning \(Tips and Tricks Cheatsheet\)](#)
- [ROC Curves and Precision-Recall Curves for Classification](#)
- [Wikipedia: Precision and Recall](#)
- [Wikipedia: F-score](#)
- [Wikipedia: Sensitivity and Specificity](#)
- [The Precision-Recall Plot Is More Informative than the ROC Plot when Evaluating Binary Classifiers on Imbalanced Datasets](#)
- [F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?](#)
- [sklearn.metrics.det\\_curve](#)
- [Wikipedia: Detection error tradeoff](#)
- [Are FAR and FRR the same as FPR and FNR, respectively?](#)
- [ROUGE and BLEU scores for NLP model evaluation](#)
- [BLUE: a Method for Automatic Evaluation of Machine Translation](#)
- [ROUGE: A Package for Automatic Evaluation of Summaries](#)
- [Sebastian Raschka's LinkedIn posts](#)
- [How do you evaluate/test accuracy of Text-to-Speech \(TTS\) models?](#)
- [Text-to-speech Synthesis by Paul Taylor](#)
- [Perplexity And Burstiness In AI And Human Writing: Two Important Concepts](#)

## Citation

If you found our work useful, please cite it as:

```
@article{Chadha2020DistilledEvaluationMetrics,  
  title   = {Evaluation Metrics, ROC-Curves and Imbalanced Datasets},  
  author  = {Chadha, Aman},  
  journal = {Distilled AI},  
  year    = {2020},
```



```
note    = {\url{https://aman.ai}}  
}
```



[www.amanchadha.com](http://www.amanchadha.com)