Logistic Regression

Avinash A S

Calculating a probability with the sigmoid function

Many problems require a probability estimate as output.

Logistic regression is an extremely efficient mechanism for calculating probabilities. Practically speaking, you can use the returned probability in either of the following two ways:

- Applied "as is." For example, if a spam-prediction model takes an email as input and outputs a value of 0.932, this implies a 93.2% probability that the email is spam.
- Converted to a **binary category** such as True or False, Spam or Not Spam.

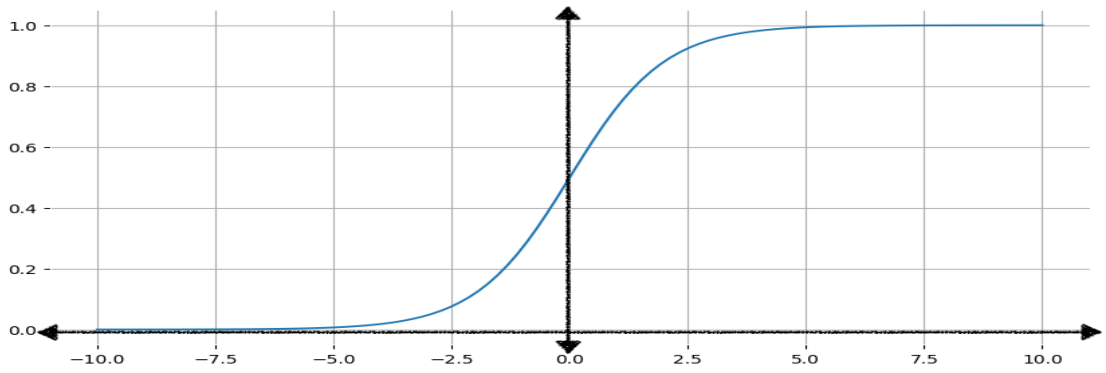
This module focuses on using logistic regression model output as-is. In the Classification module, you'll learn how to convert this output into a binary category.

Sigmoid function

You might be wondering how a logistic regression model can ensure its output represents a probability, always outputting a value between 0 and 1. As it happens, there's a family of functions called **logistic functions** whose output has those same characteristics. The standard logistic function, also known as the **sigmoid function**

(*sigmoid* means "s-shaped"), has the formula:

$$f(x) = \frac{1}{1 + e^{-x}}$$



Graph of the sigmoid function. The curve approaches 0 as *x* values decrease to negative infinity, and 1 as *x* values increase toward infinity.

As the input, x, increases, the output of the sigmoid function approaches but never reaches 1. Similarly, as the input decreases, the sigmoid function's output approaches but never reaches 0.

deeper dive into the math behind the sigmoid function

The table below shows the output values of the sigmoid function for input values in the range –7 to 7. Note how quickly the sigmoid approaches 0 for decreasing negative input values, and how quickly the sigmoid approaches 1 for increasing positive input values.

However, no matter how large or how small the input value, the output will always be greater than 0 and less than 1.

Input	Sigmoid output
-7	0.001
-6	0.002
-5	0.007
-4	0.018
-3	0.047
-2	0.119
-1	0.269
0	0.50
1	0.731
2	0.881
3	0.952
4	0.982
5	0.993
6	0.997
7	0.999

Transforming linear output using the sigmoid function

The following equation represents the linear component of a logistic regression model:

$$z = b + w_1 x_1 + w_2 x_2 + \ldots + w_N x_N$$

where:

- *z* is the output of the linear equation, also called the log odds.
- *b* is the bias.
- The *w* values are the model's learned weights.
- The *x* values are the feature values for a particular example.

To obtain the logistic regression prediction, the *z* value is then passed to the sigmoid function, yielding a value (a probability) between 0 and 1:

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

where:

- y' is the output of the logistic regression model.
- z is the linear output (as calculated in the preceding equation).

In the equation , z is referred to as the *log-odds* because if you start with the following sigmoid function

(where is the output of a logistic regression model, representing a probability): $$y' = \frac{1}{1 + e^{-z}}$$

And then solve for z: $$z = \log\left(\frac{y}{1 - y}\right)$$

Then z is defined as the log of the ratio of the probabilities of the two possible outcomes: y and $1 - y$.

Figure 2 illustrates how linear output is transformed to logistic regression output using these calculations.

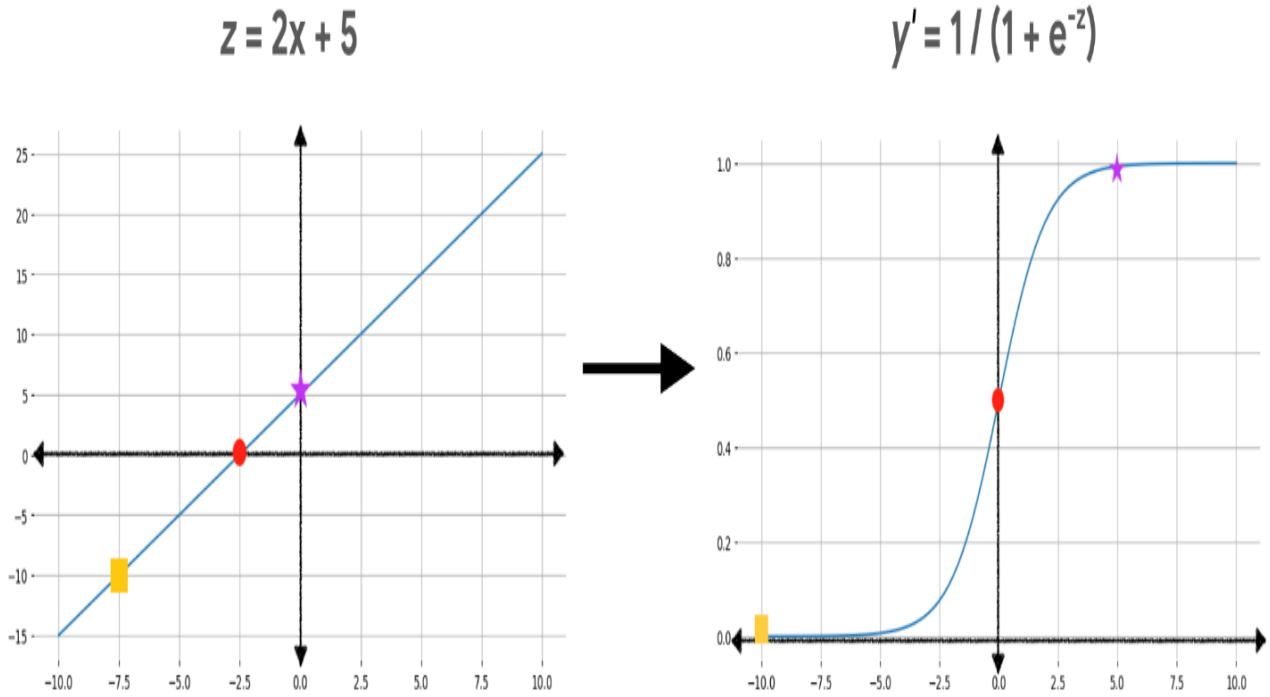


Figure 2. Left: graph of the linear function z = 2x + 5, with three points highlighted. Right: Sigmoid curve with the same three points highlighted after being transformed by the sigmoid function.

In Figure 2, a linear equation becomes input to the sigmoid function, which bends the straight line into an s-shape. Notice that the linear equation can output very big or very small values of z, but the output of the sigmoid function, y', is always between 0 and 1, exclusive. For example, the yellow square on the left

graph has a z value of –10, but the sigmoid function in the right graph maps that –10 into a y' value of 0.00004.

Logistic regression: Loss and regularization

Logistic regression models are trained using the same process as linear regression models, with two key distinctions:

- Logistic regression models use Log Loss as the loss function instead of squared loss.
- Applying regularization is critical to prevent overfitting.

The following sections discuss these two considerations in more depth.

Loss

For example: given the linear model , each time you increment the input value x1 by 1, the output value y' increases by 3.

However, the rate of change of a logistic regression model is *not* constant. As you saw in Calculating a probability, the sigmoid curve is s-shaped rather than linear. When the log-odds (z) value is closer to 0, small increases in result in much larger changes to y than z when is a large positive or negative number. The following table shows the sigmoid function's output for input values from 5 to 10, as well as the corresponding precision required to capture the differences in the results.

input	logistic output	required digits of precision
5	0.993	3
6	0.997	3
7	0.999	3
8	0.9997	4
9	0.9999	4
10	0.99998	5

If you used squared loss to calculate errors for the sigmoid function, as the output got closer and closer to 0 and 1, you would need more memory to preserve the precision needed to track these values.

Instead, the loss function for logistic regression is Log Loss. The Log Loss equation returns the logarithm of the magnitude of the change, rather than just the distance from data to prediction. Log Loss is calculated as follows:

L_1 loss

A **loss function** that calculates the absolute value of the difference between actual **label** values and the values that a **model** predicts.

For example: here's the calculation of L_1 loss for a **batch** of five **examples**:

Actual value of example	Model's predicted value	Absolute value of delta
7	6	1
5	4	1
8	11	3
4	6	2
9	8	1
		8 = L_1 loss

L_1 loss is less sensitive to outliers than L_2 loss.

The Mean Absolute Error is the average L_1 loss per example.

$$L_1 loss = \sum_{i=0}^{n} |y_i - \hat{y}_i|$$

where:

- n is the number of examples.
- y is the actual value of the label.
- y' is the value that the model predicts for y.

L_2 loss

A loss function that calculates the square of the difference between actual label values and the values that a model predicts.

For example, here's the calculation of L_2 loss for a batch of five examples

Actual value of example	Model's predicted value	Square of delta
7	6	1
5	4	1
8	11	9
4	6	4
9	8	1
		16 = L_2 loss

Due to squaring, L_2 loss amplifies the influence of outliers. That is, L_2 loss reacts more strongly to bad predictions than L_1 loss.

For example, the L_1 loss for the preceding batch would be 8 rather than 16. Notice that a single outlier accounts for 9 of the 16.

Regression models typically use L_2 loss as the loss function.

The Mean Squared Error is the average L_2 loss per example. Squared loss is another name for L_2 loss.

$$L_2 loss = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

where:

- *n* is the number of examples.
- *y* is the actual value of the label.
- *y'* is the value that the model predicts for *y*.

Regularization in logistic regression

Regularization, a mechanism for penalizing model complexity during training, is extremely important in logistic regression modeling. Without regularization, the asymptotic nature of logistic regression would keep driving loss towards 0 in cases where the model has a large number of features. Consequently, most logistic regression models use one of the following two strategies to decrease model complexity:

- L_2 regularization
- Early stopping: Limiting the number of training steps to halt training while loss is still decreasing.

In the Logistic regression module, you learned how to use the sigmoid function to convert raw model output to a value between 0 and 1 to make probabilistic prediction.

for example: predicting that a given email has a 75% chance of being spam. But what if your goal is not to output probability but a category.

predicting whether a given email is "spam" or "not spam"?

Classification is the task of predicting which of a set of classes (categories) an example belongs to. In this module, you'll learn how to convert a logistic regression model that predicts a probability into a binary classification model that predicts one of two classes. You'll also learn how to choose and calculate appropriate metrics to evaluate the quality of a classification model's predictions. Finally, you'll get a brief introduction to multi-class classification problems, which are discussed in more depth later in the course.

Thresholds and the confusion matrix

Let's say you have a logistic regression model for spam-email detection that predicts a value between 0 and 1, representing the probability that a given email is spam. A prediction of 0.50 signifies a 50% likelihood that the email is spam, a prediction of 0.75 signifies a 75% likelihood that the email is spam, and so on.

You'd like to deploy this model in an email application to filter spam into a separate mail folder. But to do so, you need to convert the model's raw numerical output (e.g., 0.75) into one of two categories: "spam" or "not spam."

To make this conversion, you choose a threshold probability, called a classification threshold. Examples with a probability above the threshold value are then assigned to the positive class, the class you are testing for (here, spam). Examples with a lower probability are assigned to the negative class, the alternative class (here, not spam).

more details on the classification threshold

Suppose the model scores one email as 0.99, predicting that email has a 99% chance of being spam, and another email as 0.51, predicting it has a 51% chance of being spam. If you set the classification threshold to 0.5, the model will classify both emails as spam. If you set the threshold to 0.95, only the email scoring 0.99 will be classified as spam.

While 0.5 might seem like an intuitive threshold, it's not a good idea if the cost of one type of wrong classification is greater than the other, or if the classes are imbalanced. If only 0.01% of emails are spam, or if misfiling legitimate emails is worse than letting spam into the inbox, labeling anything the model considers at least 50% likely to be spam as spam produces undesirable results.

Confusion matrix

The probability score is not reality, or ground truth. There are four possible outcomes for each output from a binary classifier. For the spam classifier example, if you lay out the ground truth as columns and the model's prediction as rows, the following table, called a confusion matrix, is the result:

	Actual positive	**Actual negative**
Predicted positive	**True positive (TP): A spam email correctly classified as a spam email. These are the spam messages automatically sent to the spam folder.**	**False positive (FP): A not-spam email misclassified as spam. These are the legitimate emails that wind up in the spam folder.**
Predicted negative	**False negative (FN): A spam email misclassified as not-spam. These are spam emails that aren't caught by the**	**True negative (TN): A not-spam email correctly classified as not-spam. These are the legitimate emails that are sent directly to the inbox.**

	spam filter and make their way into the inbox.	

Notice that the total in each row gives all predicted positives (TP + FP) and all predicted negatives (FN + TN), regardless of validity. The total in each column, meanwhile, gives all real positives (TP + FN) and all real negatives (FP + TN) regardless of model classification.

When the total of actual positives is not close to the total of actual negatives, the dataset is imbalanced. An instance of an imbalanced dataset might be a set of thousands of photos of clouds, where the rare cloud type you are interested in, say, volutus clouds, only appears a few times.

Effect of threshold on true and false positives and negatives

Classification: Accuracy, recall, precision, and related metrics

True and false positives and negatives are used to calculate several useful metrics for evaluating models. Which evaluation metrics are most meaningful depends on the specific model and the specific task, the cost of different misclassifications, and whether the dataset is balanced or imbalanced.

All of the metrics in this section are calculated at a single fixed threshold, and change when the threshold changes. Very often, the user tunes the threshold to optimize one of these metrics.

Accuracy

Accuracy is the proportion of all classifications that were correct, whether positive or negative. It is mathematically defined as:

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

In the spam classification example, accuracy measures the *fraction of all emails correctly classified*.

A perfect model would have zero false positives and zero false negatives and therefore an accuracy of 1.0, or 100%.

Because it incorporates all four outcomes from the confusion matrix (TP, FP, TN, FN), given a balanced dataset, with similar numbers of examples in both classes, accuracy can serve as a coarse-grained measure of model quality. For this reason, it is often the default evaluation metric used for generic or unspecified models carrying out generic or unspecified tasks.

However, when the dataset is imbalanced, or where one kind of mistake (FN or FP) is more costly than the other, which is the case in most real-world applications, it's better to optimize for one of the other metrics instead.

For heavily imbalanced datasets, where one class appears very rarely, say 1% of the time, a model that predicts negative 100% of the time would score 99% on accuracy, despite being useless.

Recall, or true positive rate

The true positive rate (TPR), or the proportion of all actual positives that were classified correctly as positives, is also known as recall.

Recall is mathematically defined as:

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

False negatives are actual positives that were misclassified as negatives, which is why they appear in the denominator. In the spam classification example, recall measures the *fraction of spam emails that were correctly classified as spam*. This is why another name for recall is probability of detection: it answers the question "What fraction of spam emails are detected by this model?"

A hypothetical perfect model would have zero false negatives and therefore a recall (TPR) of 1.0, which is to say, a 100% detection rate.

In an imbalanced dataset where the number of actual positives is very low, recall is a more meaningful metric than accuracy because it measures the ability of the model to correctly identify all positive instances. For applications like disease prediction, correctly identifying the positive cases is crucial. A false negative typically has more serious consequences than a false positive. For a concrete example comparing recall and accuracy metrics, see the notes in the definition of recall.

False positive rate

The false positive rate (FPR) is the proportion of all actual negatives that were classified *incorrectly* as positives, also known as the probability of false alarm. It is mathematically defined as:

$$\text{FPR} = \frac{\text{incorrectly classified actual negatives}}{\text{all actual negatives}} = \frac{FP}{FP + TN}$$

False positives are actual negatives that were misclassified, which is why they appear in the denominator. In the spam classification example, FPR measures the *fraction of legitimate emails that were incorrectly classified as spam,* or the model's rate of false alarms.

A perfect model would have zero false positives and therefore a FPR of 0.0, which is to say, a 0% false alarm rate.

In an imbalanced dataset where the number of actual negatives is very, very low, say 1-2 examples in total, FPR is less meaningful and less useful as a metric.

Precision

Precision is the proportion of all the model's positive classifications that are actually positive. It is mathematically defined as:

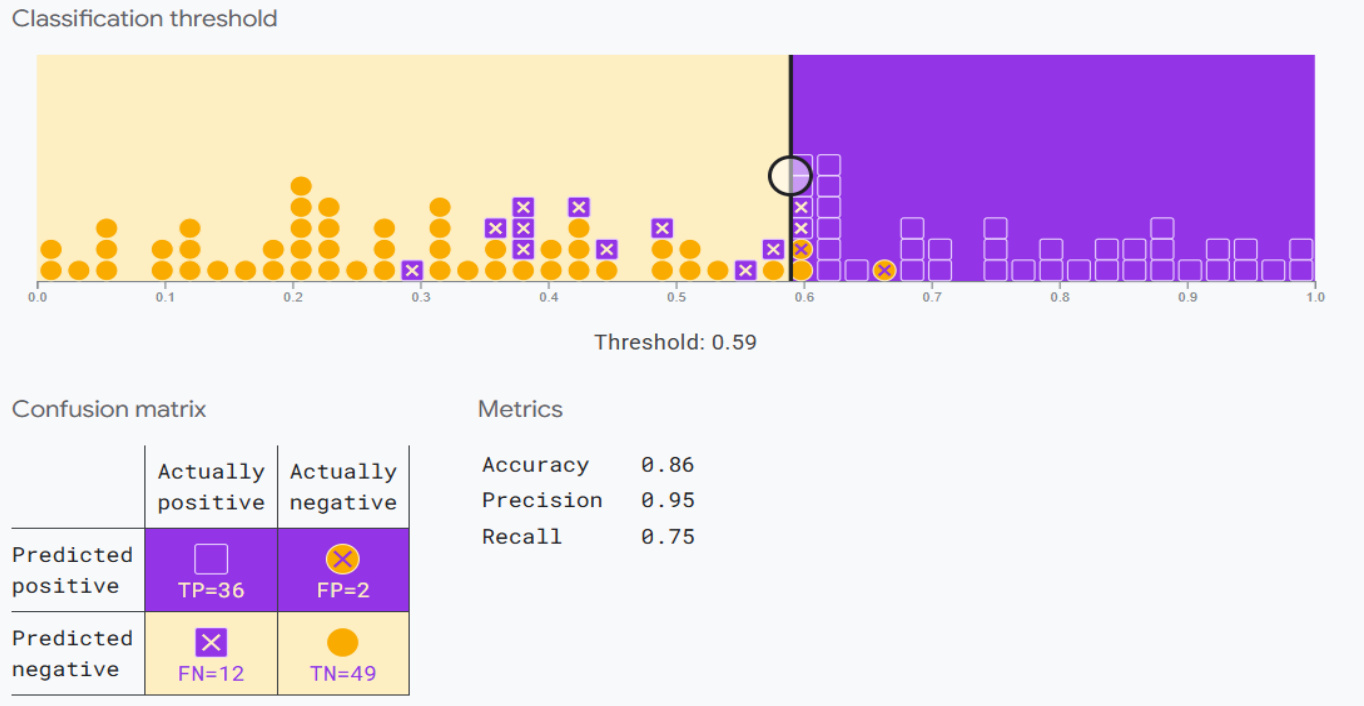
$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

In the spam classification example, precision measures the *fraction of emails classified as spam that were actually spam*.

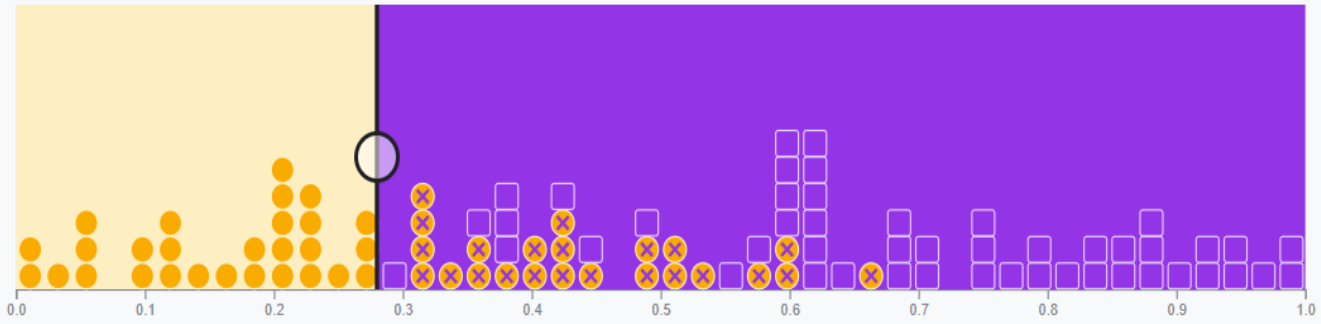
A hypothetical perfect model would have zero false positives and therefore a precision of 1.0.

In an imbalanced dataset where the number of actual positives is very, very low, say 1-2 examples in total, precision is less meaningful and less useful as a metric.

Precision improves as false positives decrease, while recall improves when false negatives decrease. But as seen in the previous section, increasing the classification threshold tends to decrease the number of false positives and increase the number of false negatives, while decreasing the threshold has the opposite effects. As a result, precision and recall often show an inverse relationship, where improving one of them worsens the other.



Classification threshold



Threshold: 0.28

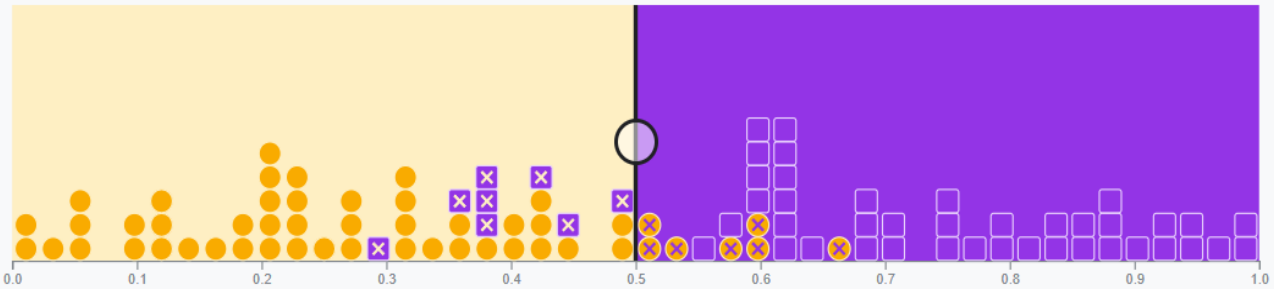
Confusion matrix

	Actually positive	Actually negative
Predicted positive	TP=48	FP=23
Predicted negative	FN=0	TN=28

Metrics

Accuracy	0.77
Precision	0.68
Recall	1.00

Classification threshold



Threshold: 0.50

Confusion matrix

	Actually positive	Actually negative
Predicted positive	TP=40	FP=7
Predicted negative	FN=8	TN=44

Metrics

Accuracy	0.85
Precision	0.85
Recall	0.83

Choice of metric and tradeoffs

The metric(s) you choose to prioritize when evaluating the model and choosing a threshold depend on the costs, benefits, and risks of the specific problem. In the spam classification example, it often makes sense to prioritize recall, nabbing all the spam emails, or precision, trying to ensure that spam-labeled emails are in fact spam, or some balance of the two, above some minimum accuracy level.

Metric	Guidance
Accuracy	Use as a rough indicator of model training progress/convergence for balanced datasets. For model performance, use only in combination with other metrics. Avoid for imbalanced datasets. Consider using another metric.
Recall (True positive rate)	Use when false negatives are more expensive than false positives.
False positive rate	Use when false positives are more expensive than false negatives.
Precision	Use when it's very important for positive predictions to be accurate.

Classification: ROC and AUC

The previous section presented a set of model metrics, all calculated at a single classification threshold value. But if you want to evaluate a model's quality across all possible thresholds, you need different tools.

Receiver-operating characteristic curve (ROC)

The ROC curve is a visual representation of model performance across all thresholds. The long version of the name, receiver operating characteristic, is a holdover from WWII radar detection.

The ROC curve is drawn by calculating the true positive rate (TPR) and false positive rate (FPR) at every possible threshold (in practice, at selected intervals), then graphing TPR over FPR. A perfect model, which at some threshold has a TPR of 1.0 and a FPR of 0.0, can be represented by either a point at (0, 1) if all other thresholds are ignored, or by the following:

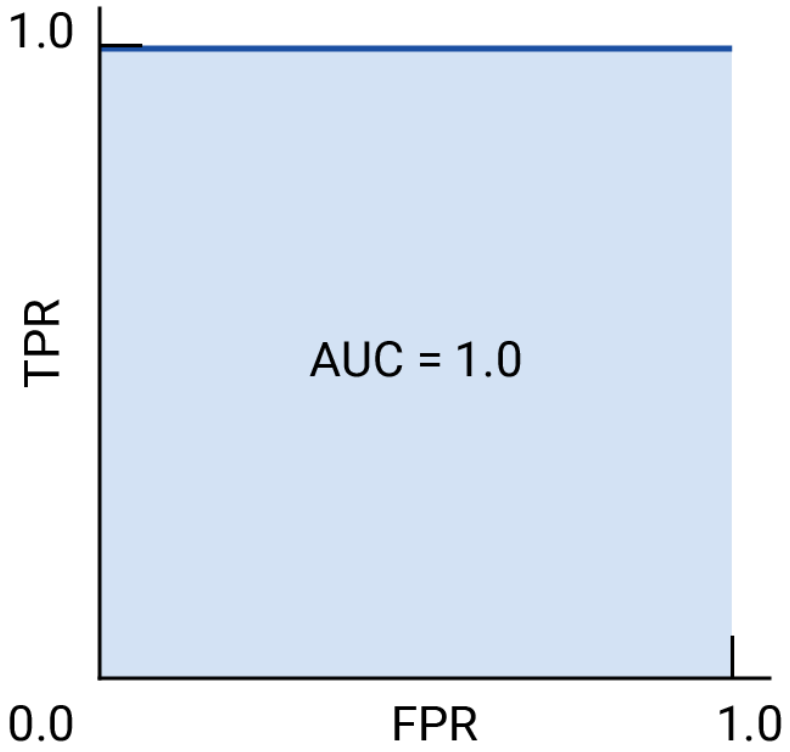


Figure 1. ROC and AUC of a hypothetical perfect model.

Area under the curve (AUC)

The area under the ROC curve (AUC) represents the probability that the model, if given a randomly chosen positive and negative example, will rank the positive higher than the negative.

The perfect model above, containing a square with sides of length 1, has an area under the curve (AUC) of 1.0. This means there is a 100% probability that the model will correctly rank a randomly chosen positive example higher than a randomly chosen negative example. In other words, looking at the spread of data points below, AUC gives the probability that the model will place a randomly chosen square to the right of a randomly chosen circle, independent of where the threshold is set.

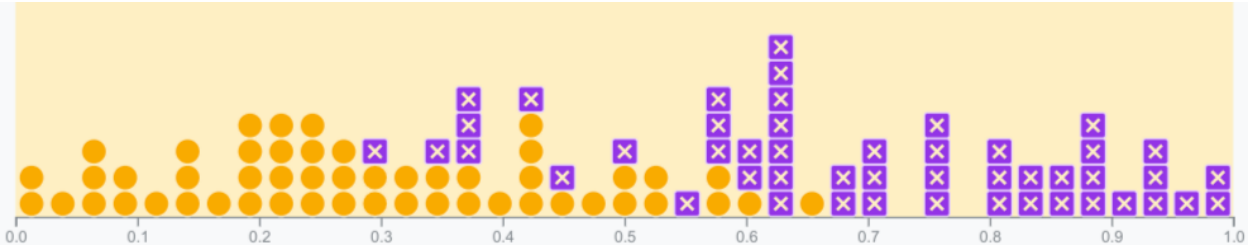


Figure 2. A spread of predictions for a binary classification model. AUC is the chance a randomly chosen square is positioned to the right of a randomly chosen circle.

In more concrete terms, a spam classifier with AUC of 1.0 always assigns a random spam email a higher probability of being spam than a random legitimate email. The actual classification of each email depends on the threshold that you choose.

For a binary classifier, a model that does exactly as well as random guesses or coin flips has a ROC that is a diagonal line from (0,0) to (1,1). The AUC is 0.5, representing a 50% probability of correctly ranking a random positive and negative example.

In the spam classifier example, a spam classifier with AUC of 0.5 assigns a random spam email a higher probability of being spam than a random legitimate email only half the time.

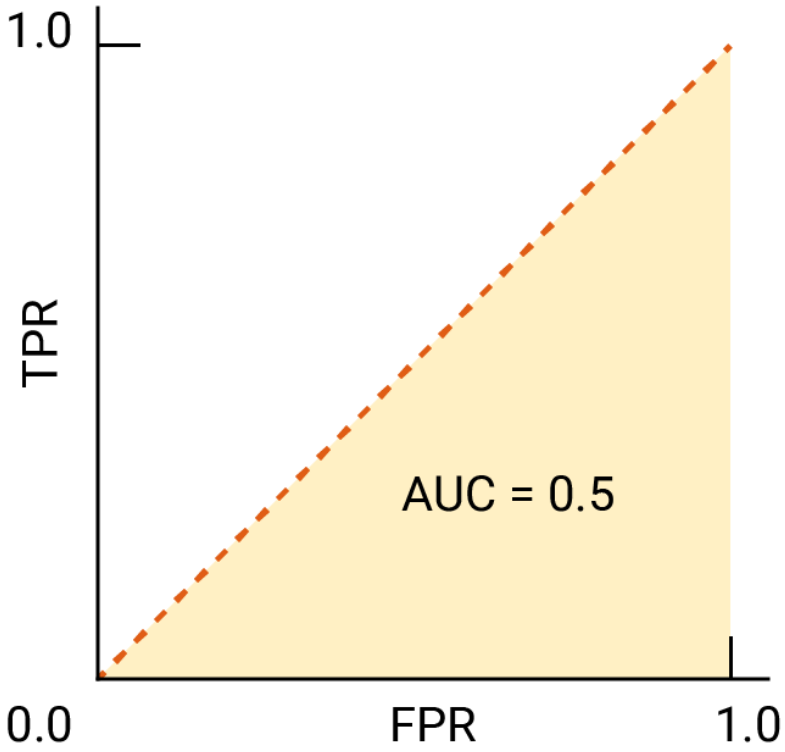
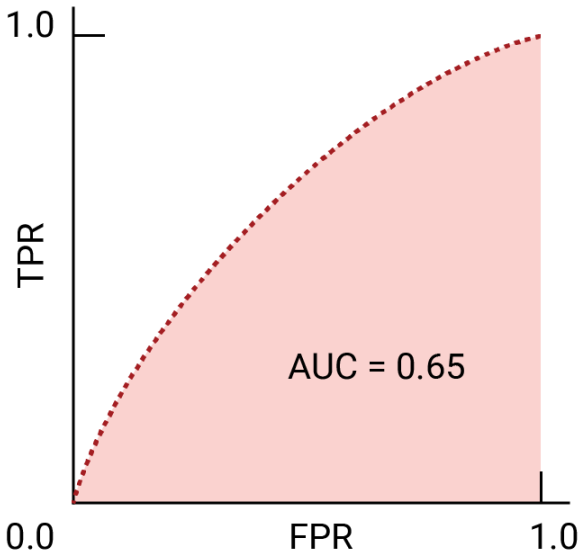


Figure 3. ROC and AUC of completely random guesses.

AUC and ROC for choosing model and threshold

AUC is a useful measure for comparing the performance of two different models, as long as the dataset is roughly balanced. The model with greater area under the curve is generally the better one.



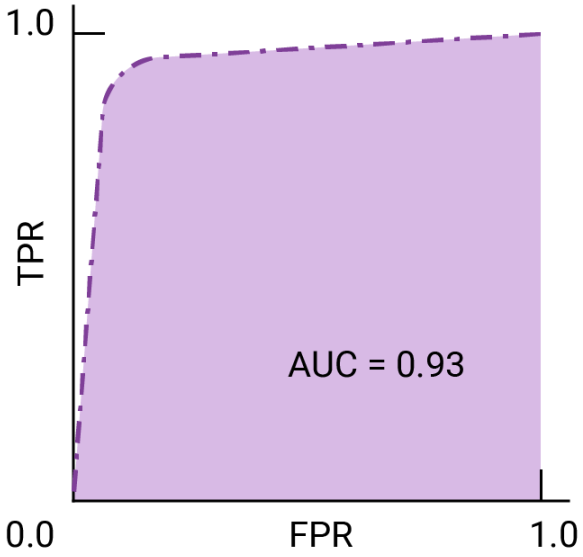


Figure 4. ROC and AUC of two hypothetical models. The curve on the right, with a greater AUC, represents the better of the two models.

The points on a ROC curve closest to (0,1) represent a range of the best-performing thresholds for the given model. As discussed in the Thresholds, Confusion matrix and Choice of metric and tradeoffs sections, the threshold you choose depends on which metric is most important to the specific use case. Consider the points A, B, and C in the following diagram, each representing a threshold:

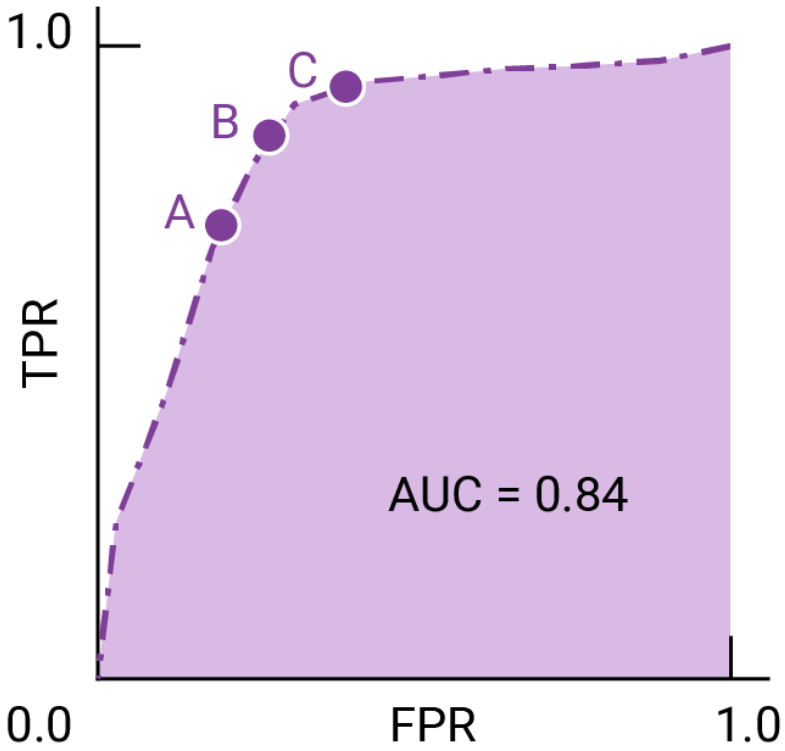
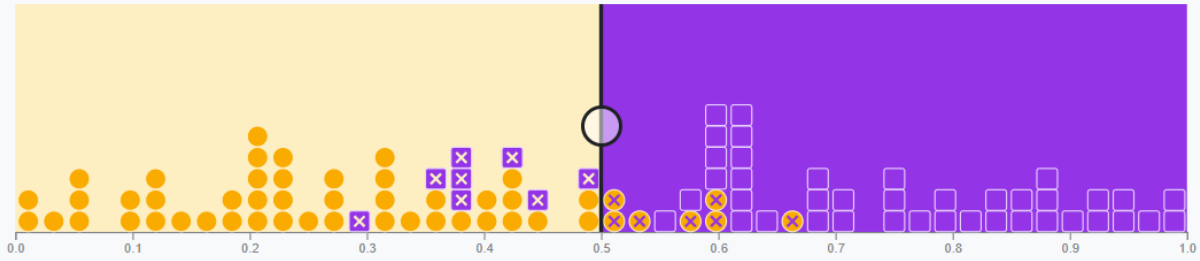


Figure 5. Three labeled points representing thresholds.

If false positives (false alarms) are highly costly, it may make sense to choose a threshold that gives a lower FPR, like the one at point A, even if TPR is reduced. Conversely, if false positives are cheap and false negatives (missed true positives) highly costly, the threshold for point C, which maximizes TPR, may be preferable. If the costs are roughly equivalent, point B may offer the best balance between TPR and FPR.

Classification threshold



Threshold: 0.50

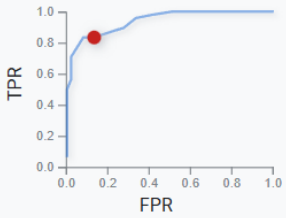
Confusion matrix

	Actually positive	Actually negative
Predicted positive	TP=40	FP=7
Predicted negative	FN=8	TN=44

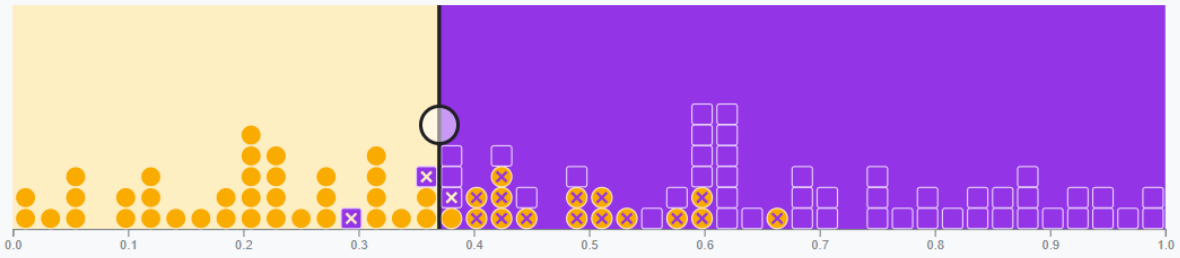
Metrics

Accuracy	0.85
Precision	0.85
Recall	0.83

ROC curve



Classification threshold



Threshold: 0.37

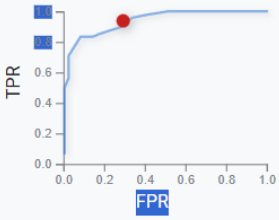
Confusion matrix

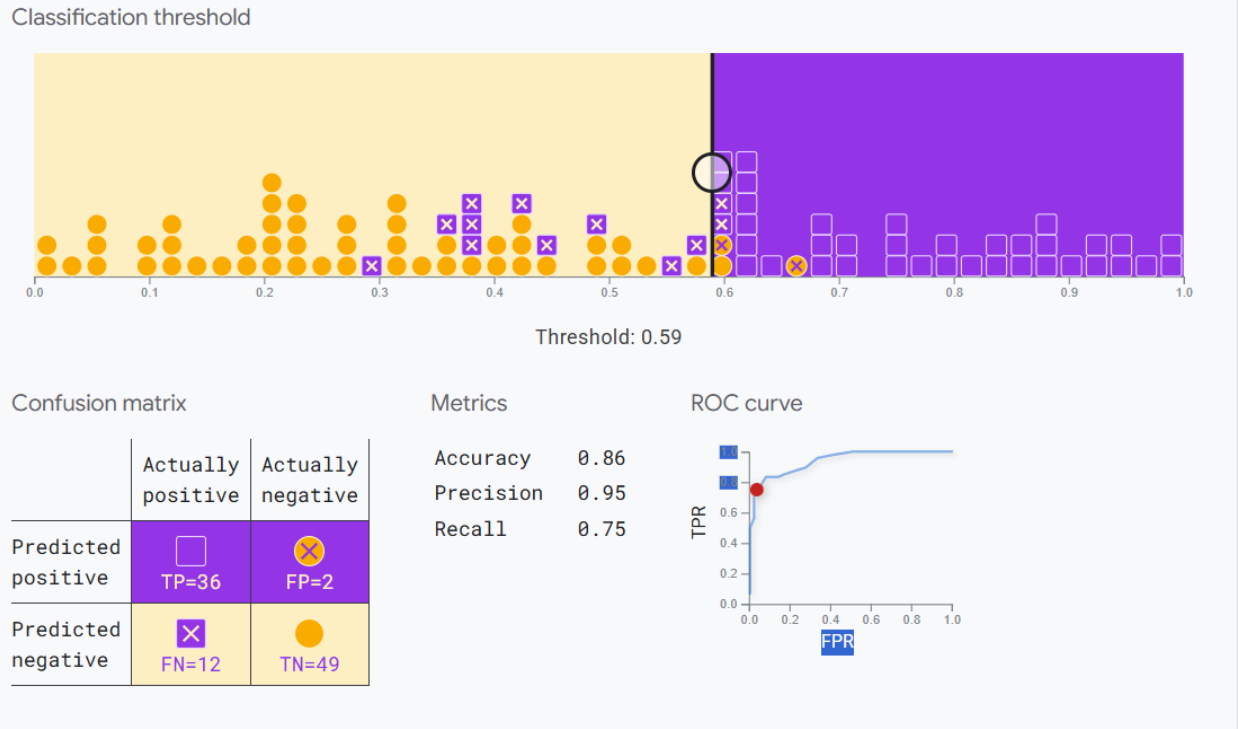
	Actually positive	Actually negative
Predicted positive	TP=45	FP=15
Predicted negative	FN=3	TN=36

Metrics

Accuracy	0.82
Precision	0.75
Recall	0.94

ROC curve





Classification: Prediction bias

bookmark_border

As mentioned in the Linear regression module, calculating prediction bias is a quick check that can flag issues with the model or training data early on.

Prediction bias is the difference between the mean of a model's predictions and the mean of ground-truth labels in the data. A model trained on a dataset where 5% of the emails are spam should predict, on average, that 5% of the emails it classifies are spam. In other words, the mean of the labels in the ground-truth dataset is 0.05, and the mean of the model's predictions should also be 0.05. If this is the case, the model has zero prediction bias. Of course, the model might still have other problems.

If the model instead predicts 50% of the time that an email is spam, then something is wrong with the training dataset, the new dataset the model is applied to, or with the model itself. Any significant difference between the two means suggests that the model has some prediction bias.

Prediction bias can be caused by:

- **Biases or noise in the data, including biased sampling for the training set**
- **Too-strong regularization, meaning that the model was oversimplified and lost some necessary complexity**
- **Bugs in the model training pipeline**
- **The set of features provided to the model being insufficient for the task**

Classification: Multi-class classification

- Multi-class classification can be treated as an extension of **binary classification** to more than two classes. If each example can only be assigned to one class, then the classification problem can be handled as a binary classification problem, where one class contains one of the multiple classes, and the other class contains all the other classes put together. The process can then be repeated for each of the original classes.
- For example, in a three-class multi-class classification problem, where you're classifying examples with the labels **A**, **B**, and **C**, you could turn the problem into two separate binary classification problems. First, you might create a binary classifier that categorizes examples using the label **A+B** and the label **C**. Then, you could create a second binary classifier that reclassifies the examples that are labeled **A+B** using the label **A** and the label **B**.
- An example of a multi-class problem is a handwriting classifier that takes an image of a handwritten digit and decides which digit, 0-9, is represented.
- If class membership isn't exclusive, which is to say, an example can be assigned to multiple classes, this is known as a *multi-label* classification problem.