

Binary classification


12 minutes

Classification, like regression, is a *supervised* machine learning technique; and therefore follows the same iterative process of training, validating, and evaluating models. Instead of calculating numeric values like a regression model, the algorithms used to train classification models calculate *probability* values for class assignment and the evaluation metrics used to assess model performance compare the predicted classes to the actual classes.

Binary classification algorithms are used to train a model that predicts one of two possible labels for a single class. Essentially, predicting **true** or **false**. In most real scenarios, the data observations used to train and validate the model consist of multiple feature (x) values and a y value that is either 1 or 0.

Example - binary classification

To understand how binary classification works, let's look at a simplified example that uses a single feature (x) to predict whether the label y is 1 or 0. In this example, we'll use the blood glucose level of a patient to predict whether or not the patient has diabetes. Here's the data with which we'll train the model:

 Expand table

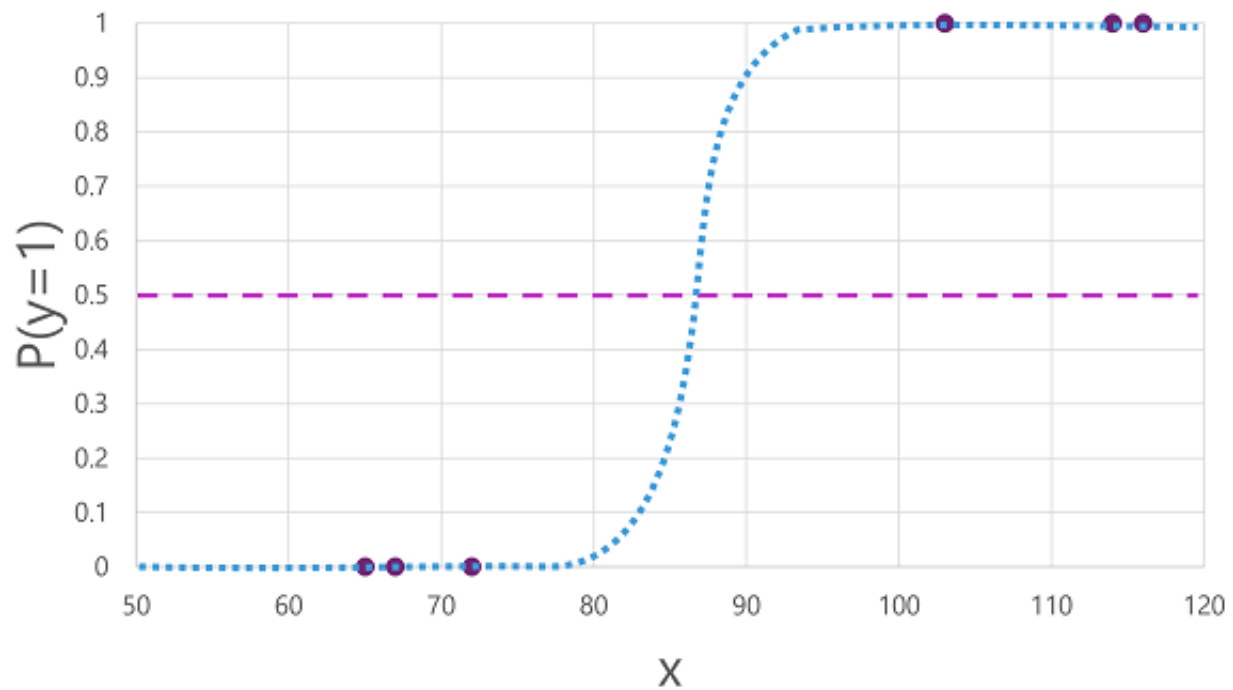


Blood glucose (x)	Diabetic? (y)
67	0
103	1
114	1
72	0
116	1
65	0

Training a binary classification model

To train the model, we'll use an algorithm to fit the training data to a function that calculates the *probability* of the class label being *true* (in other words, that the patient has diabetes). Probability is measured as a value between 0.0 and 1.0, such that the *total* probability for *all* possible classes is 1.0. So for example, if the probability of a patient having diabetes is 0.7, then there's a corresponding probability of 0.3 that the patient isn't diabetic.

There are many algorithms that can be used for binary classification, such as *logistic regression*, which derives a *sigmoid* (S-shaped) function with values between 0.0 and 1.0, like this:



ⓘ Note

Despite its name, in machine learning *logistic regression* is used for classification, not regression. The important point is the *logistic* nature of the function it produces, which describes an S-shaped curve between a lower and upper value (0.0 and 1.0 when used for binary classification).

The function produced by the algorithm describes the probability of **y** being true ($y=1$) for a given value of **x**. Mathematically, you can express the function like this:

$$f(x) = P(y=1 \mid x)$$

For three of the six observations in the training data, we know that **y** is definitely *true*, so the probability for those observations that $y=1$ is **1.0** and for the other three, we know that **y** is definitely *false*, so the probability that $y=1$ is **0.0**. The S-shaped curve describes the probability distribution so that plotting a value of **x** on the line identifies the corresponding probability that **y** is 1.

The diagram also includes a horizontal line to indicate the *threshold* at which a model

based on this function will predict *true* (1) or *false* (0). The threshold lies at the mid-point for y ($P(y) = 0.5$). For any values at this point or above, the model will predict *true* (1); while for any values below this point it will predict *false* (0). For example, for a patient with a blood glucose level of 90, the function would result in a probability value of 0.9. Since 0.9 is higher than the threshold of 0.5, the model would predict *true* (1) - in other words, the patient is predicted to have diabetes.

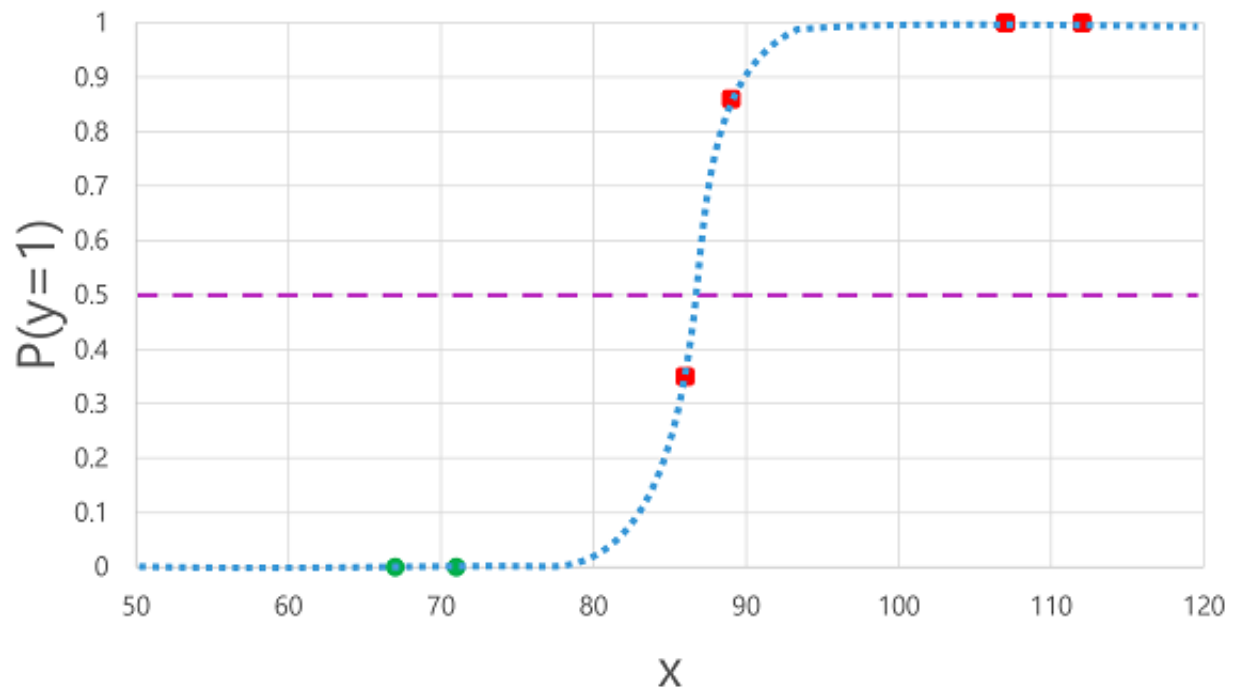
Evaluating a binary classification model

As with regression, when training a binary classification model you hold back a random subset of data with which to validate the trained model. Let's assume we held back the following data to validate our diabetes classifier:

 Expand table

Blood glucose (x)	Diabetic? (y)
66	0
107	1
112	1
71	0
87	1
89	1

Applying the logistic function we derived previously to the x values results in the following plot.



Based on whether the probability calculated by the function is above or below the threshold, the model generates a predicted label of 1 or 0 for each observation. We can then compare the *predicted* class labels (\hat{y}) to the *actual* class labels (y), as shown here:

[Expand table](#)

Blood glucose (x)	Actual diabetes diagnosis (y)	Predicted diabetes diagnosis (\hat{y})
66	0	0
107	1	1
112	1	1
71	0	0

87	1	0
89	1	1

Binary classification evaluation metrics

The first step in calculating evaluation metrics for a binary classification model is usually to create a matrix of the number of correct and incorrect predictions for each possible class label:

		\hat{y}	
		0	1
y	0	2	0
	1	1	3

This visualization is called a *confusion matrix*, and it shows the prediction totals where:

- $\hat{y}=0$ and $y=0$: *True negatives* (TN)
- $\hat{y}=1$ and $y=0$: *False positives* (FP)
- $\hat{y}=0$ and $y=1$: *False negatives* (FN)
- $\hat{y}=1$ and $y=1$: *True positives* (TP)

The arrangement of the confusion matrix is such that correct (*true*) predictions are shown in a diagonal line from top-left to bottom-right. Often, color-intensity is used to indicate the number of predictions in each cell, so a quick glance at a model that predicts well should reveal a deeply shaded diagonal trend.

Accuracy

The simplest metric you can calculate from the confusion matrix is *accuracy* - the proportion of predictions that the model got right. Accuracy is calculated as:

$$(TN+TP) \div (TN+FN+FP+TP)$$

In the case of our diabetes example, the calculation is:

$$(2+3) \div (2+1+0+3)$$

$$= 5 \div 6$$

$$= 0.83$$

So for our validation data, the diabetes classification model produced correct predictions 83% of the time.

Accuracy might initially seem like a good metric to evaluate a model, but consider this. Suppose 11% of the population has diabetes. You could create a model that always predicts **0**, and it would achieve an accuracy of 89%, even though it makes no real attempt to differentiate between patients by evaluating their features. What we really need is a deeper understanding of how the model performs at predicting **1** for positive cases and **0** for negative cases.

Recall

Recall is a metric that measures the proportion of positive cases that the model identified correctly. In other words, compared to the number of patients who *have* diabetes, how many did the model *predict* to have diabetes?

The formula for recall is:

$$TP \div (TP+FN)$$

For our diabetes example:

$$3 \div (3+1)$$

$$= 3 \div 4$$

$$= 0.75$$

So our model correctly identified 75% of patients who have diabetes as having diabetes.

Precision

Precision is a similar metric to recall, but measures the proportion of predicted positive cases where the true label is actually positive. In other words, what proportion of the patients *predicted* by the model to have diabetes actually *have* diabetes?

The formula for precision is:

$$TP \div (TP + FP)$$

For our diabetes example:

$$3 \div (3 + 0)$$

$$= 3 \div 3$$

$$= 1.0$$

So 100% of the patients predicted by our model to have diabetes do in fact have diabetes.

F1-score

F1-score is an overall metric that combined recall and precision. The formula for F1-score is:

$$(2 \times Precision \times Recall) \div (Precision + Recall)$$

For our diabetes example:

$$(2 \times 1.0 \times 0.75) \div (1.0 + 0.75)$$

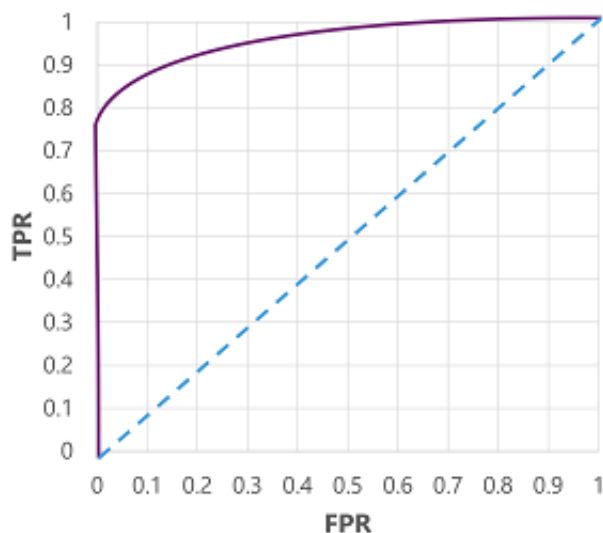
$$= 1.5 \div 1.75$$

$$= 0.86$$

Area Under the Curve (AUC)

Another name for recall is the *true positive rate* (TPR), and there's an equivalent metric called the *false positive rate* (FPR) that is calculated as $\text{FP} \div (\text{FP} + \text{TN})$. We already know that the TPR for our model when using a threshold of 0.5 is 0.75, and we can use the formula for FPR to calculate a value of $0 \div 2 = 0$.

Of course, if we were to change the threshold above which the model predicts *true* (1), it would affect the number of positive and negative predictions; and therefore change the TPR and FPR metrics. These metrics are often used to evaluate a model by plotting a *received operator characteristic* (ROC) curve that compares the TPR and FPR for every possible threshold value between 0.0 and 1.0:



The ROC curve for a perfect model would go straight up the TPR axis on the left and then across the FPR axis at the top. Since the plot area for the curve measures 1x1, the area under this perfect curve would be 1.0 (meaning that the model is correct 100% of the time). In contrast, a diagonal line from the bottom-left to the top-right represents the results that would be achieved by randomly guessing a binary label; producing an area under the curve of 0.5. In other words, given two possible class labels, you could

reasonably expect to guess correctly 50% of the time.

In the case of our diabetes model, the curve above is produced, and the **area under the curve** (AUC) metric is 0.875. Since the AUC is higher than 0.5, we can conclude the model performs better at predicting whether or not a patient has diabetes than randomly guessing.

Next unit: Multiclass classification

[< Previous](#)[Next >](#)