# CHAPTER 5: Classification Models

## Overview

In this chapter, you will explore different types of classification models. You will gain hands-on experience of using TensorFlow to build binary, multi-class, and multi-label classifiers. Finally, you will learn the concepts of model evaluation and how you can use different metrics to assess the performance of a model.

By the end of this chapter, you will have a good understanding of what classification models are and how programming with TensorFlow works.

## Introduction

In the previous chapter, you learned about regression problems where the target variable is continuous. A continuous variable can take any value between a minimum and maximum value. You learned how to train such models with TensorFlow.

In this chapter, you will look at another type of supervised learning problem called classification, where the target variable is discrete — meaning it can only take a finite number of values. In industry, you will most likely encounter such projects where variables are aggregated into groups such as product tiers, or classes of users, customers, or salary ranges. The objective of a classifier is to learn the patterns from the data and predict the right class for observation.

For instance, in the case of a loan provider, a classification model will try to predict whether a customer is most likely to default in the coming year based on their profile and financial position. This outcome can only take two possible values (yes or no), which is a binary classification. Another classifier model could predict the ratings from 1 to 5 of a new movie for a user given their previous ratings and the information about this movie. When the outcome can be more than two possible values, you are dealing with a multi-class classification. Finally, there is a third type of classifier called multi-label where the model will predict more than a class. For example, a model will analyze an input image and predict whether there is a cat, a dog, or a mouse in the image. In such a case, the model will predict three different binary outputs (or labels).

You will go through each type of classifier in this chapter, detail their specificities, and explore how to measure the performance of these models.

## Binary Classification

As mentioned previously, binary classification refers to a type of supervised learning where the target variable can only take two possible values (or classes) such as true/false or yes/no. For instance, in the medical industry, you may want to predict whether a patient is more likely to have a disease based on their personal information such as age, height, weight, and/or medical measurements. Similarly, in marketing, advertisers might utilize similar information to optimize email campaigns.

Machine learning algorithms such as the random forest classifier, support vector classifier, or logistic regression work well for classification. Neural networks can also achieve good results for binary classification. It is extremely easy to turn a regression model such as those in the previous chapter into a binary classifier. There are only two key changes required: the activation function for the last layer and the loss function.

## Logistic Regression

**Logistic regression** is one of the most popular algorithms for dealing with binary classification. As its name implies, it is an extension of the linear regression algorithm. A linear regression model predicts an output that can take an infinite number of values within a range. For logistic regression, you want your model to predict values between 0 and 1. The value 0 usually corresponds to false (or no) while the value 1 refers to true (or yes).

In other terms, the output of logistic regression will be the probability of it being true. For example, if the output is 0.3, you can say there is a probability of 30% that the result should be true (or yes). But as there are only two possible values, this will also mean there is a probability of 70% (100% – 30%) of having the outcome of false (or no):

Timeline

Description automatically generated

Figure 5.1: Output of logistic regression

Now that you know what the output of logistic regression is, you just need to find a function that can transform an input value that is continuous into a value between 0 and 1. Luckily, such a mathematical function does exist, and it is called the sigmoid function. The formula for this function is as follows:

A picture containing text, clock

Description automatically generated

Figure 5.2: Formula of the sigmoid function

 corresponds to the exponential function applied to x.

The exponential function ranges from 0 to positive infinity. So, if x has a value close to positive infinite, the value of sigmoid will tend to 1. On the other hand, if x is very close to negative infinite, then the value of sigmoid will tend to 0:

A picture containing chart

Description automatically generated

Figure 5.3: Curve of the sigmoid function

So, applying the sigmoid function on the output of a linear regression model turns it into logistic regression. The same logic holds for neural networks: if you apply the sigmoid function on a perceptron model (linear regression), you will get a binary classifier. To do so, you just need to specify sigmoid as the activation function of the last fully connected layer of a perceptron model. In TensorFlow, you specify the activation parameter as:

Graphical user interface, text, application

Description automatically generated

The preceding code snippet shows how to define a fully connected layer with a single unit that can output any value and apply the sigmoid activation function to it. The result will then be within 0 and 1. Now that you know how to modify a neural network's regression model to turn it into a binary classifier, you need to specify the relevant loss function.

## Binary Cross-Entropy

In the previous section, you learned how to turn a linear regression model into a binary classifier. With neural networks, it is as simple as adding sigmoid as the activation function for the last fully connected layer. But there is another consideration that will impact the training of this model: the choice of the loss function.

For linear regression, the most frequently used loss functions are **mean squared error** and **mean absolute error** as seen in *Chapter 4*, *Regression and Classification Models*. These functions will calculate the difference between the predicted and the actual values, and the neural network model will update all its weights accordingly during backpropagation. For a binary classification, the typical loss function is **binary cross-entropy** (also called **log loss**). The formula for this function is as follows:

A picture containing logo

Description automatically generated

Figure 5.4: Formula of binary cross-entropy

represents the actual value for the observation i.

represents the predicted probability for the observation i.

N represents the total number of observations.

This formula looks quite complicated, but its logic is quite simple. Consider the following example of a single observation: the actual value is 1 and the predicted probability is 0.8. If the preceding formula is applied, the result will be as follows:

Formula

Notice that the right-hand side of the equation is approximately zero:

Formula

So, the loss value will be very small as the predicted value is very close to the actual one.

Now consider another example where the actual value is 0 and the predicted probability is 0.99. The result will be as follows:

Formula

Formula

The loss will be high in this case as the prediction is very different from the actual value.

The **binary cross-entropy function** is a good fit for assessing the difference between predicted and actual values for a binary classification. TensorFlow provides a class called BinaryCrossentropy that computes this loss:

Text

Description automatically generated

## Binary Classification Architecture

The architecture for binary classifiers is extremely similar to that of linear regression as seen in Chapter 4, Regression and Classification Models. It is composed of an input layer that reads each observation of the input dataset, an output layer responsible for predicting the response variable, and some hidden layers that learn the patterns leading to the correct predictions. The following diagram shows an example of such an architecture:

Diagram

Description automatically generated

Figure 5.5: Architecture of the binary classifier

The only difference compared to linear regression is the output, which is a probability value between 0 and 1. This probability indicates the likelihood of the occurrence for one of the two possible values. As seen previously, this is achieved using the sigmoid activation function and binary cross-entropy for backpropagation.

Now that you have seen all the elements to build a binary classifier, you can put this into practice with an exercise.

## Exercise 5.01: Building a Logistic Regression Model

In this exercise, you will build and train a logistic regression model in TensorFlow that will predict whether the signal shows the presence of some object, or just empty air.

You will be working on The Ionosphere dataset contains features obtained from radar signals focused on the ionosphere layer of the Earth's atmosphere.

Note

The training dataset can be accessed [here](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/datasets/ion.csv)

1. Open a new Jupyter notebook.
2. Import the pandas and sklearn libraries.

Shape

Description automatically generated with medium confidence

1. Create a variable called data\_url that contains the URL to the dateset:



1. Load the dataset into a DataFrame() function called data using read\_csv( ) method, provide the URL to the CSV file. We have dropped the start column to which contain row numbers. Print the first five rows of the DataFrame using head() method:

Text

Description automatically generated

The expected output will be as follows:

Table

Description automatically generated

Figure 5.6: The first five rows of the dataset

You can see that the dataset contains 35 columns, and they are all numeric, except the target variable. Note also that the target variable (column 35) contains two different values: good and bad. As you will train a logistic regression model, the possible values should be 0 and 1. You will need to replace the “good” values with 1 and “bad” values with 0.

1. Let’s encode our target variables with 0 and 1

Chart

Description automatically generated with medium confidence

1. Extract the target variable (column 35) using pop() method and save it in variable y and remaining independent variables as X.

Text

Description automatically generated with low confidence

1. Now, let’s split our dataset into train and test sets. Where our test set contains 30% of the data.

Graphical user interface, text

Description automatically generated

If you print the shape of the split sets, you will get the following output.

Text

Description automatically generated

Text, letter

Description automatically generated

1. Import TensorFlow library and use tf as the alias:

Rectangle

Description automatically generated with low confidence

1. Set the seed for TensorFlow as 8, using tf.random.set\_seed() to get reproducible results:

A picture containing chart

Description automatically generated

1. Instantiate a sequential model using tf.keras.Sequential() and store it in a variable called model:

A picture containing graphical user interface

Description automatically generated

1. Import the Dense() class from tensorflow.keras.layers:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 512 units with Dense() and specify ReLu as the activation function and the input shape as (34,), which corresponds to the number of features from the dataset. Save it in a variable called fc1:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 512 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc2:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 128 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc3:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 128 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc4:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 1 units with Dense() and specify sigmoid as the activation function. Save it in a variable called fc5:

A picture containing rectangle

Description automatically generated

1. Sequentially add all five fully connected layers to the model using add() method:

Text, application

Description automatically generated

1. Print the summary of the model using summary() method:

Text

Description automatically generated with medium confidence

The expected output will be as follows:

Table

Description automatically generated

Figure 5.7: Summary of the model architecture

The preceding output shows that there are five layers in your model (as expected) and displays the number of parameters at each layer. For example, the first layer contains 17,920 parameters, and the total number of parameters for this model is 362,881. All these parameters will be trained while fitting the model.

1. Instantiate a BinaryCrossentropy() function from tf.keras.losses and save it in a variable called loss:

A picture containing text

Description automatically generated

1. Instantiate Adam() from tf.keras.optimizers with 0.001 as the learning rate and save it in a variable called optimizer:

A picture containing rectangle

Description automatically generated

1. Compile the model using the compile() function and specify the optimizer and loss you just created in previous steps:

A picture containing shape

Description automatically generated

1. Start the model training process using fit() method on the training set for five epochs:

A picture containing text

Description automatically generated

The expected output will be as follows:

A picture containing table

Description automatically generated

Figure 5.8: Logs of the training process

The preceding output shows the logs of each epoch during the training of the model. Note that it took around 3 seconds to process a single epoch and the loss value decreased from 0.5599 (first epoch) to 0.0889 (fifth epoch), so the model is slowly improving its performance by reducing the binary cross-entropy loss.

1. Predict the results of the test set using predict() method. Save it in a variable called preds and display its first five values:

Graphical user interface, application

Description automatically generated

The expected output will be as follows:

Text

Description automatically generated

Figure 5.9: Predictions of the first five rows of the test set

The preceding output shows the probability of each prediction. Each value below 0.5 will be classified as 0 (first and last observation in this output) and all values greater than or equal to 0.5 will be 1 (second to fourth observations).

1. Display the first five true labels of the test set:



The expect output will be as follows:

A picture containing text

Description automatically generated

Figure 5.10: Ture labels of the first five rows of the test set

Comparing this output with the model predictions on the first five rows of the test set, there are some incorrect values: the third prediction (index 2) should be a value of 0 and the last one should be 0. So, out of these five observations, your binary classifiers made two mistakes.

1. Finally, we save our model weights to the h5 file.

Text

Description automatically generated

In the section ahead, you will see how to properly evaluate the performance of a model with different metrics.

## Metrics for Classifiers

In the previous section, you learned how to train a binary classifier to predict the right output: either 0 or 1. In *Exercise 5.01*, *Building a Logistic Regression Model*, you looked at a few samples to assess the performance of the models that were built. Usually, you would evaluate a model not just on a small subset but on the whole dataset using a performance metric such as accuracy or F1 score.

## Accuracy and Null Accuracy

One of the most widely used metrics for classification problems is accuracy. Its formula is quite simple:

Figure 5.14: Formula of the accuracy metric


Figure 5.11: Formula of the accuracy metric

The maximum value for accuracy is 1, which means the model correctly predicts 100% of the cases. Its minimum value is 0, where the model can't predict any case correctly.

For a binary classifier, the number of correct predictions is the number of observations with a value of 0 or 1 as the correctly predicted value:

Figure 5.15: Formula of the accuracy metric for a binary classifier


Figure 5.12: Formula of the accuracy metric for a binary classifier

Say you are assessing the performance of two different binary classifiers predicting the outcome on 10,000 observations on the test set. The first model correctly predicted 5,000 instances of value 0 and 3,000 instances of value 1. Its accuracy score will be as follows:

Figure 5.16: Formula for the accuracy of model1


Figure 5.13: Formula for the accuracy of model1

The second model correctly predicted the value 0 for 500 cases and the value 1 for 1,500 observations. Its accuracy score will be as follows:

Figure 5.17: Formula for the accuracy of model2


Figure 5.14: Formula for the accuracy of model2

The first model predicts accurately 80% of the time, while the second model is only 20% accurate. In this case, you can say that model 1 is better than model 2.

Even though 0.8 is usually a relatively good score, this does not necessarily mean your model is performing well. For instance, say your dataset contains 9,000 cases of value 0 and 1,000 cases of value 1. A very simple model that always predicts value 0 will achieve an accuracy score of 0.9. In this case, the first model is performing even less well than this extremely simple model. This characteristic of such a model that always predicts the most frequent value of a dataset is called the **null accuracy**. It is used as a baseline to compare with other trained models. In the preceding example, the null accuracy is 0.9 since the simple model predicts 0, which is correct 90% of the time.

Note

The accuracy and null accuracy metrics are not specific to binary classification but can also be applied to other types of classification.

TensorFlow provides a class, tf.keras.metrics.Accuracy, that can calculate the accuracy score from tensors. This class has a method called update\_state() that takes two tensors as input parameters and will compute the accuracy score between them. You can access this score by calling the result() method. The output result will be a tensor. You can use the numpy() method to convert it into a NumPy array. Here is an example of how to calculate the accuracy score:

Text

Description automatically generated with medium confidence

This will result in the following accuracy score:

A picture containing table

Description automatically generated

Note

TensorFlow doesn't provide a class for the null accuracy metric, but you can easily compute it using Accuracy() and provide a tensor with only 1 (or 0) as the predictions.

## Precision, Recall, and the F1 Score

In the previous section, you learned how to use the accuracy metric to assess the performance of a model and compare it against a baseline called the null accuracy. The accuracy score is widely used as it is well known to non-technical audiences, but it does have some limitations. Consider the following example.

Figure 5.18: Example of model predictions versus actual values


Figure 5.15: Example of model predictions versus actual values

This model achieves an accuracy score of 0.981 Diagram

Description automatically generated, which is quite high. But if this model is used to predict whether a person has a disease, it will only predict correctly in a single case. It incorrectly predicted in nine cases that these people are not sick while they actually have the given disease. At the same time, it incorrectly predicted sickness for 10 people who were actually healthy. This model's performance, then, is clearly unsatisfactory. Unfortunately, the accuracy score is simply an overall score, and it doesn't tell you where the model is performing badly.

Luckily, other metrics provide a better assessment of a model, such as precision, recall, or F1 score. All three of these metrics have the same range of values as the accuracy score: 1 is the perfect score, wherein all observations are predicted correctly, and 0 is the worst, wherein there is no correct prediction at all.

But before looking at them, you need to be familiar with the following definitions:

* **True Positive (TP)**: All the observations where the actual value and the corresponding prediction are both true
* **True Negative (TN)**: All the observations where the actual value and the corresponding prediction are both false
* **False Positive (FP)**: All the observations where the prediction is true, but the values are actually false
* **False Negative (FN)**: All the observations where the prediction is false, but the values are actually true

Taking the same example as *Figure 5.14*, you will get the following:

A picture containing chart

Description automatically generated

This is seen in the following table:

Figure 5.19: Example of TP, TN, FP, and FN


Figure 5.16: Example of TP, TN, FP, and FN

The precision score is a metric that assesses whether a model has predicted a lot of FPs. Its formula is as follows:

Figure 5.20: Formula of precision


Figure 5.17: Formula of precision

In the preceding example, the precision score will be A picture containing text

Description automatically generated. You can see this model is making a lot of mistakes and has predicted a lot of FPs compared to the actual TP.

Recall is used to assess the number of FNs compared to TPs. Its formula is as follows:

Figure 5.21: Formula of recall


Figure 5.18: Formula of recall

In the preceding example, the recall score will be Text

Description automatically generated with medium confidence. With this metric, you can see that the model is not performing well and is predicting a lot of FNs.

Finally, the F1 score is a metric that combines both precision and recall (it is the harmonic mean of precision and recall). Its formula is as follows:

Figure 5.22: Formula for the F1 score


Figure 5.19: Formula for the F1 score

Taking the same example as the preceding, the F1 score will be

Formula

The model has achieved an F1 score of 0.095, which is very different from its accuracy score of 0.981. So, the F1 score is a good performance metric when you want to emphasize the incorrect predictions—the score considers the number of FNs and FPs in the score, as well as the TPs and TNs.

Note

As with accuracy, precision, and recall performance metrics, the F1 score can also be applied to other types of classification.

You can easily calculate precision and recall with TensorFlow by using the respective classes of Precision() and Recall():

Text, letter

Description automatically generated

This result in the following output:

A picture containing shape

Description automatically generated

Figure 5.20: Precision and recall scores of the provided example

Note

TensorFlow doesn't provide a class to calculate the F1 score, but this can easily be done by creating a custom metric.

## Confusion Matrices

A confusion matrix is not a performance metric *per se*, but more a graphical tool used to visualize the predictions of a model against the actual values. You have actually already seen an example of this in the previous section

A confusion matrix will show all the possible values of the predictions on one axis (for example, the horizontal axis) and the actual values on the other axis (the vertical axis). At the intersection of each combination of predicted and actual values, you will record the number of observations that fall under this case.

For a binary classification, the confusion matrix will look like the following:

Figure 5.24: Confusion matrix for a binary classification


Figure 5.21: Confusion matrix for a binary classification

The ideal situation will be that all the values sit on the diagonal of this matrix. This will mean your model is correctly predicting all possible values. All values outside of this diagonal are where your model made some mistakes.

Note

Confusion matrices can also be used for multi-class classification and are not specific to binary classification only.

Run the code below to see the confusion matrix:

Text

Description automatically generated

This will display the following output:

Text, letter

Description automatically generated

## Exercise 5.02: Classification Evaluation Metrics

In this exercise, you will reuse the same logistic regression model as in Exercise 5.01, Building a Logistic Regression Model, and assess its performance by looking at different performance metrics: accuracy, precision, recall, and F1 score.

Now, run the following instructions:

1. Open a new Jupyter notebook.
2. Import the pandas and sklearn libraries.

Shape

Description automatically generated with medium confidence

1. Create a variable called data\_url that contains the URL to the dateset:



1. Load the dataset into a DataFrame() function called data using read\_csv( ) method, provide the URL to the CSV file. We have dropped the start column to which contain row numbers. Print the first five rows of the DataFrame using head() method:

Text

Description automatically generated

The expected output will be as follows:

Table

Description automatically generated

Figure 5.6: The first five rows of the dataset

You can see that the dataset contains 35 columns, and they are all numeric, except the target variable. Note also that the target variable (column 35) contains two different values: good and bad. As you will train a logistic regression model, the possible values should be 0 and 1. You will need to replace the “good” values with 1 and “bad” values with 0.

1. Let’s encode our target variables with 0 and 1

Chart

Description automatically generated with medium confidence

1. Extract the target variable (column 35) using pop() method and save it in variable y and remaining independent variables as X.

Text

Description automatically generated with low confidence

1. Now, let’s split our dataset into train and test sets. Where our test set contains 30% of the data.

Graphical user interface, text

Description automatically generated

If you print the shape of the split sets, you will get the following output.

Text

Description automatically generated

Text, letter

Description automatically generated

1. Impot tensorFlow package

Text

Description automatically generated with medium confidence

1. Load the model and print the model summary using the summary() method:

A picture containing application

Description automatically generated

Text

Description automatically generated with low confidence

You should get the following output:

Table

Description automatically generated

Figure 5.22: Summary of the model

The preceding output shows the same architecture as the model from Exercise 5.01, Building a Logistic Regression Model.

1. Predict the results of the test set using predict() method. Save it in a variable called preds\_proba and display its first five values:

A picture containing graphical user interface

Description automatically generated

The expected output will be as follows:

Chart

Description automatically generated with medium confidence

Figure 5.23: Predicted probabilities of the test set

The outputs are the predicted probabilities of being 1 (or true) for each observation. You need to convert these probabilities into 0 and 1 only. To do so, you will need to consider all cases with a probability greater than or equal to 0.5 to be 1 (or true), and 0 (or false) for the records with a probability lower than 0.5.

1. Convert the predicted probabilities into 1 when the probability is greater than or equal to 0.5, and 0 when below 0.5. Save the results in a variable called preds and print its first five rows:

Text

Description automatically generated with low confidence

The expected output will be as follows:

Text

Description automatically generated with medium confidence

1. Import Accuracy, Precision, and Recall from tensorflow.keras.metrics:

A picture containing chart

Description automatically generated

1. Instantiate Accuracy, Precision, and Recall objects and save them in variables called acc, pres, and rec, respectively:

Graphical user interface, text, application

Description automatically generated

1. Calculate the accuracy score on the test set with the update\_state(), result(), and numpy() methods. Save the results in a variable called acc\_results and print its content:

Graphical user interface, text, application

Description automatically generated

The expected output will be as follows:



This model achieved an accuracy score of 0.97.

1. Calculate the precision score on the test set with the update\_state(), result(), and numpy() methods. Save the results in a variable called prec\_results and print its content:

Graphical user interface, text, application

Description automatically generated

The expected output will be as follows:

Text

Description automatically generated with low confidence

This model achieved a precision score of 0.97.

1. Calculate the recall score on the test set with the update\_state(), result(), and numpy() methods. Save the results in a variable called rec\_results and print its content:

Graphical user interface, text, application

Description automatically generated

The expected output will be as follows:

Text

Description automatically generated

This model achieved a recall score of 0.97.

1. Calculate the F1 score by applying the formula shown in the previous section. Save the result in a variable called f1 and print its content:

Graphical user interface, text

Description automatically generated

The expected output will be as follows:



Overall, the model has achieved excellent score close to 0.98 for all four different metrics: accuracy, precision, recall and F1 score. So, this model makes almost correct predictions.

In the section ahead, you will be looking at expanding classification to more than two possible values with multi-class classification.

## Multi-Class Classification

With binary classification, you were limited to dealing with target variables that can only take two possible values: 0 and 1 (false or true). Multi-class classification can be seen as an extension of this and allows the target variable to have more than two values (or you can say binary classification is just a subset of multi-class classification). For instance, a model that predicts different levels of disease severity for a patient or another one that classifies users into different groups based on their past shopping behaviors will be multi-class classifiers.

In the next section, you will dive into the softmax function, which is used for multi-class classification.

## The Softmax Function

Binary classifiers require a specific activation function for the last fully connected layer of a neural network, which is sigmoid. The activation function specific to multi-class classifiers is different. It is softmax. Its formula is as follows:

Figure 5.29: Formula of softmax function


Figure 5.24: Formula of softmax function

Formula 1corresponds to the predicted value for class i.

Formula 1corresponds to the predicted value for class j.

This formula will be applied to each possible value of the target variable. If you have 10 possible values, then this activation function will calculate 10 different softmax values.

Note that softmax exponentiates the predicted values on both the numerator and the denominator. The reason behind this is that the exponential function magnifies small changes between predicted values and makes probabilities lie closer to 0 or 1 for the purpose of interpreting the resulting output. For instance, exp(2) = 7.39 while exp(2.2) = 9.03. So, if two classes have predicted values close to each other, the difference between their exponentiated values will be much bigger and therefore it will be easier to select the higher one.

The result of the softmax function is between 0 and 1 as the method divides the value for one class by the sum of all the classes. So, the actual output of a softmax function is the probability of the relevant class being the final prediction:

Diagram

Description automatically generated

In the preceding example, the target variable has five different values, and the softmax function transforms them into probabilities. The first class (0) is the one with the highest probability, and this will be the final prediction.

## Categorical Cross-Entropy

Multi-class classification also requires a specific loss function that is different from the binary cross-entropy for binary classifiers. For multi-class classification, the loss function is categorical cross-entropy. Its formula is as follows:

Figure 5.31: Formula of categorical cross-entropy


Figure 5.25: Formula of categorical cross-entropy

Formularepresents the probability of the actual value for the observation i to be of class j.

Formula 1represents the predicted probability for the observation i to be of class j.

TensorFlow provides two different classes for this loss function: CategoricalCrossentropy() and SparseCategoricalCrossentropy():

Text

Description automatically generated

The difference between them lies in the format of the target variable. If the actual values are stored as a one-hot encoding representing the actual class, then you will need to use CategoricalCrossentropy(). On the other hand, if the response variable is stored as integers for representing the actual classes, you will have to use SparseCategoricalCrossentropy():

Diagram

Description automatically generated

Figure 5.26: Loss function to be used depending on the format of the target variable

The output of a multi-class model will be a vector containing probabilities for each class of the target variable, such as the following:

Graphical user interface, text, application, chat or text message

Description automatically generated

The first value (0.54) corresponds to the probability of having the class at index 0, 0.016 is the probability of the class at index 1, while 0.09 corresponds to the probability for the class of index 2, and so on.

In order to get the final prediction (that is, the class with the highest probability), you need to use the argmax() function, which will look at all the values from a vector, find the maximum one, and return the index associated with it:

Graphical user interface

Description automatically generated with low confidence

This will display the following output:



In the preceding example, the final prediction is class 0, which corresponds to the vector index with the highest probability (0.54).

## Multi-Class Classification Architecture

The architecture for a multi-class classifier is very similar to logistic regression, except that the last layer will contain more units. Each of them corresponds to a class of the target variable. For instance, if you are building a model that takes as input a vector of size 6 and predicts a response with three different values with a single hidden layer, its architecture will look like the following:

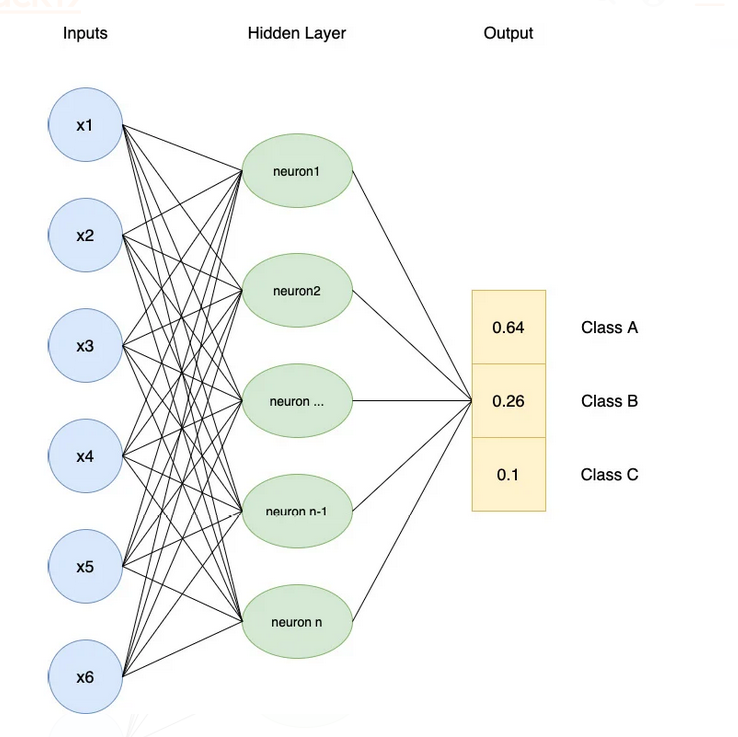


Figure 5.27: Architecture of a multi-class classifier

The softmax activation function at the last layer provides a probability of occurrence for each of the possible classes: A, B, and C. These probabilities are dependent on each other as there should be only one class predicted at the end. If class A is more likely to be the prediction (as in the preceding example), then the probabilities for the remaining classes (B and C) should be lower. Note that the sum of all the class probabilities equals 1. So, they are indeed dependent on one another.

Now that you know all the building blocks, you can build a multi-class classifier in the following exercise.

## Exercise 5.03: Building a Multi-Class Model

In this exercise, you will build and train a multi-class classifier in TensorFLow that will to correctly classify the type of surface defects in stainless steel plates, with six types of possible defects (plus “other”). The input vector was made up of 27 indicators that describe the geometric shape of the defect and its outline.

Note

The dataset can be found [here](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/datasets/faults.csv)

Perform the following steps to complete the exercise:

1. Open a new Jupyter notebook.
2. Import the pandas and sklearn libraries:

A picture containing rectangle

Description automatically generated

1. Create a variable called data\_url that contains the URL to the dateset:



1. Load the dataset into a DataFrame() function called data using read\_csv( ) method, provide the URL to the CSV file. Print the first five rows of the DataFrame using head() method:

Text

Description automatically generated with low confidence

The expected output will be as follows:

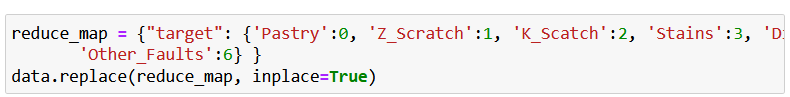
Table

Description automatically generated

Figure 5.28: The first five rows of the dataset

You can see that the dataset contains 35 columns, and they are all numeric, except the target variable. Note also that the target variable (column 35) contains two different values: good and bad. As you will train a logistic regression model, the possible values should be 0 and 1. You will need to replace the “good” values with 1 and “bad” values with 0.

1. Let’s encode our target variables with 0 to 7 values.



1. Extract the target variable (column 35) using pop() method and save it in variable y and remaining independent variables as X. Before extracting lets drop the null values.

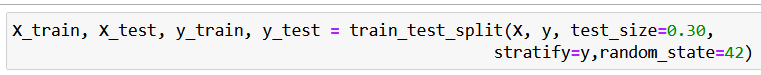
Text

Description automatically generated with low confidence

A picture containing text

Description automatically generated

1. Now, let’s split our dataset into train and test sets. Where our test set contains 30% of the data.



If you print the shape of the split sets, you will get the following output.

Text, chat or text message

Description automatically generated

Text, whiteboard

Description automatically generated

1. Import TensorFlow library and use tf as the alias:

Rectangle

Description automatically generated with low confidence

1. Set the seed for TensorFlow as 8, using tf.random.set\_seed() to get reproducible results:

A picture containing chart

Description automatically generated

1. Instantiate a sequential model using tf.keras.Sequential() and store it in a variable called model:

A picture containing graphical user interface

Description automatically generated

1. Import the Dense() class from tensorflow.keras.layers:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 512 units with Dense() and specify ReLu as the activation function and the input shape as (27,), which corresponds to the number of features from the dataset. Save it in a variable called fc1:



1. Create a fully connected layer of 512 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc2:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 128 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc3:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 128 units with Dense() and specify ReLu as the activation function. Save it in a variable called fc4:

A picture containing text

Description automatically generated

1. Create a fully connected layer of 7 units with Dense() and specify sigmoid as the activation function. Save it in a variable called fc5:



1. Sequentially add all five fully connected layers to the model using add() method:

Text, application

Description automatically generated

1. Print the summary of the model using summary() method:

Text

Description automatically generated with medium confidence

The expected output will be as follows:

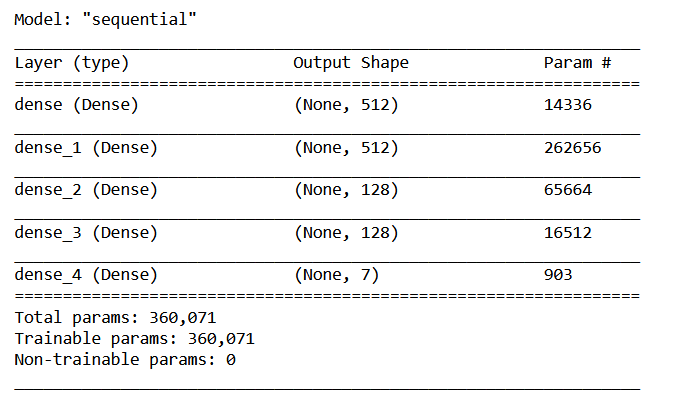


Figure 5.29: Summary of the model architecture

The preceding output shows that there are five layers in your model (as expected) and displays the number of parameters at each layer. For example, the first layer contains 17,920 parameters, and the total number of parameters for this model is 362,881. All these parameters will be trained while fitting the model.

1. Instantiate a SparseCategoricalCrossentropy() function from tf.keras.losses and save it in a variable called loss:

A picture containing graphical user interface

Description automatically generated

1. Instantiate Adam() from tf.keras.optimizers with 0.001 as the learning rate and save it in a variable called optimizer:

A picture containing rectangle

Description automatically generated

1. Compile the model using the compile() function and specify the optimizer and loss you just created in previous steps:

Text

Description automatically generated with medium confidence

1. Start the model training process using fit() method on the training set for fifty epochs:



The expected output will be as follows:

Table

Description automatically generated

Figure 5.30: Logs of the training process

The preceding output shows the logs of each epoch during the training of the model. Note that it took around 3 seconds to process a single epoch and the loss value decreased from 49464.62 (first epoch) to 1.833 (48) then loss started increasing to 32.32 at 50th epoch. Model seems overfitted.

1. Evaluate the performance of the model on the test set using the evaluate() method:



The expected output will be as follows:

A picture containing diagram

Description automatically generated

Figure 5. 31: Performance of the model on the test set.

In this exercise, you learned how to build and train a multi-class classifier to predict an outcome composed of eight different classes. Your model achieved an accuracy score close to 0.45 on both the training and test sets, which is not up to the mark. This implies that your model predicts half of the observations correct and rest as incorrect.

Let’s consolidate your learning in the following activity.

## Activity 5.01: Building a Multi-Class with TensorFlow

In this activity, you are tasked with building and training a multi-class classifier that will predict Motor Failure Time for given configuration of blades.

In this dataset, we have three independent variables pctid, x, y, and z and dependent variable wconfid

The dataset can be accessed from [here](https://raw.githubusercontent.com/fenago/tf/main/Chapter5-Classification_Models/datasets/accelerometer.csv)

1. Load the data with read\_csv() from pandas.
2. Extract the target variable with pop() method from pandas.
3. Split the data into training and test sets.
4. Build the multi-class classifier with five fully connected layers of 512, 512, 128, 128, and 26 units, respectively.
5. Train this model on the training set.
6. Evaluate its performance on the test set with evaluate() method from TensorFlow.
7. Print the confusion matrix with confusion\_matrix() from TensorFlow.

The expected output is as follows:

Text, letter

Description automatically generated

Figure 5.32: Confusion matrix of the test set

Note

The solution to this activity can be found via [this link](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/Activity%205.01-%20Building%20a%20Multi-Class%20Model%20with%20TensorFlow.ipynb)

## Activity 5.02: Building a Handwritten Digit Classification model with TensorFlow

In this activity, you are tasked with building and training a multi-label classifier that will classify the 10 digits label of numeric values from 0 to 9 from images. In this dataset, we have 784-pixel feature variable which are independent variable and 1 target label variable with values 0 to 9. The goal of this model is to determine which of the 10 digit each observation belongs to.

The training dataset can be accessed [here](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/datasets/MNIST_train.csv?raw=true)

The testing dataset can be accessed [here](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/datasets/MNIST_test.csv?raw=true)

1. Load the data with read\_csv() from pandas.
2. Extract the target variable with pop() method from pandas.
3. Split the data into training and test sets.
4. Build the multi-class classifier with five fully connected layers of 512, 512, 128, 128, and 26 units, respectively.
5. Train this model on the training set.
6. Evaluate its performance on the test set with evaluate() method from TensorFlow.

Diagram

Description automatically generated

Figure 5.33: Expected output of Activity 5.02

Note

The solution to this activity can be found via [this link](https://github.com/fenago/tf/blob/main/Chapter5-Classification_Models/Activity%205.02-%20Building%20a%20Handwritten%20Digit%20Classification%20model%20with%20TensorFlow.ipynb).

## Multi-Label Classification

Multi-label classification is another type of classification where you predict not only one target variable as in binary or multi-class classification, but several response variables at the same time. For instance, you can predict multiple outputs for the different objects present in an image (for instance, a model will predict whether there is a cat, a man, and a car in a given picture) or you can predict multiple topics for an article (such as whether the article is about the economy, international news, and manufacturing).

Implementing a multi-label classification with neural networks is extremely easy, and you have already learned everything required to build one. In TensorFlow, a multi-label classifier's architecture will look the same as for multi-class, with a final output layer with multiple units corresponding to the number of target variables you want to predict. But instead of using softmax as the activation function and categorical cross-entropy as the loss function, you will use sigmoid and binary cross-entropy as the activation and loss functions, respectively.

The sigmoid function will predict the probability of occurrence for each target variable:

Diagram

Description automatically generated

Figure 5.40: Architecture of the multi-label classifier

In the preceding example, you have three target variables and each of them has a probability of occurrence that is independent of the others (their sum will not equal 1). This model predicts that targets 2 and 3 are very likely to be the outputs for this observation.

Conceptually, multi-label classification combines several logistic regression models. They will share the same parameters (weights and biases) but with independent binary outputs. The last layer of the example of a multi-class classifier in TensorFlow will look like this:

Text

Description automatically generated with low confidence

The loss function to be used will be binary cross-entropy:

Text

Description automatically generated with medium confidence

## Summary

You started your journey in this chapter with an introduction to classification models and their differences compared with regression models. You learned that the target variable for classifiers can only contain a limited number of possible values.

You then explored binary classification, wherein the response variable can only be from two possible values: 0 or 1. You uncovered the specificities for building a logistic regression model with TensorFlow using the sigmoid activation function and binary cross-entropy as the loss function, and you built your own binary classifier.

After this, you went through the different performance metrics that can be used to assess the performance of classifier models. You practiced calculating accuracy, precision, recall, and F1 scores with TensorFlow, and also plotted a confusion matrix, which is a visual tool to see where the model made correct and incorrect predictions.

Then you dove into the topic of multi-class classification. The difference between such models and binary classifiers is that their response variables can take more than two possible values. You looked at the softmax activation function and the categorical cross-entropy loss function, which are used for training such models in TensorFlow.

Finally, in the last section, you learned about multi-label classification, wherein the output can be multiple classes at the same time. In TensorFlow, such models can be easily built by constructing an architecture similar to multi-class classification.

In the next chapter, you will learn how to prevent model overfitting by applying some regularization techniques, which will help models to better generalize unseen data.