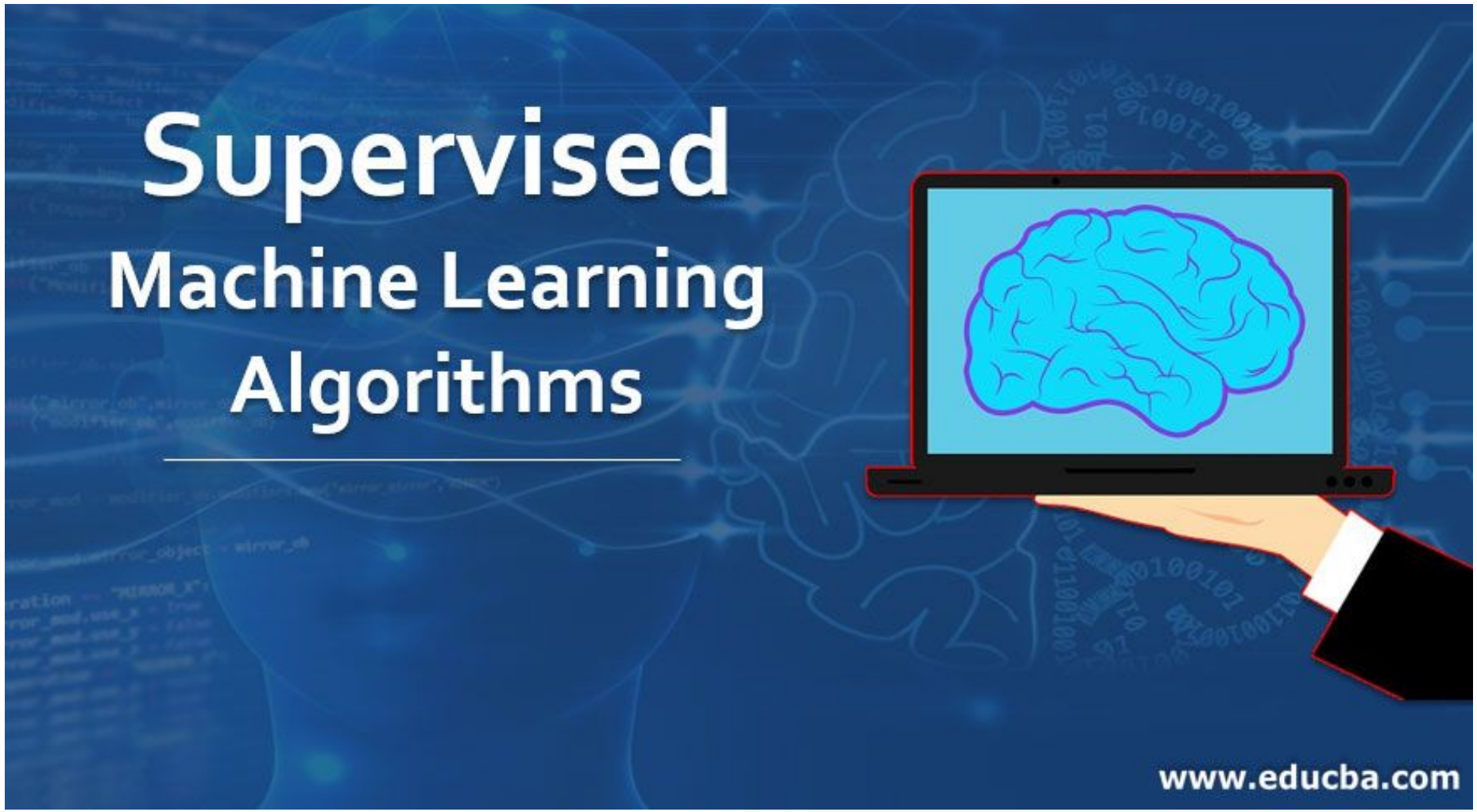

Supervised
Machine Learning
Algorithms
www.educba.com

Last lecture reminder

We learned about:

- Logistic Regression - Introduction
- Probability, Likelihood and Odds
- Logistic Regression - Theory
- Logistic Regression - Example

Logistic Regression - Python Example

Let’s execute Logistic Regression using Python and Scikit-Learn.

For the example we will use the **‘hearing_test.csv’** file.

This file represent a hearing test results and contain data about the tester ‘age’ and ‘physical_score’.

The result for each tester will be 1 (passed the test) or 0 (not pass the test).

So we clearly can see that this data is binary categorical data with 2 features - ‘age’ and ‘physical_score’.

What we want is to perform logistic regression on the existing data in order to predict test results according to future unseen data.

Logistic Regression - Python Example

First, let's examine the data itself and see what we can learn from it before starting training our model.

```
In [49]: df = pd.read_csv('/Users/ben.meir/Downloads/UNZIP_FOR_NOTEBOOKS_FINAL/DATA/hearing_test.csv')
         df.head()
```

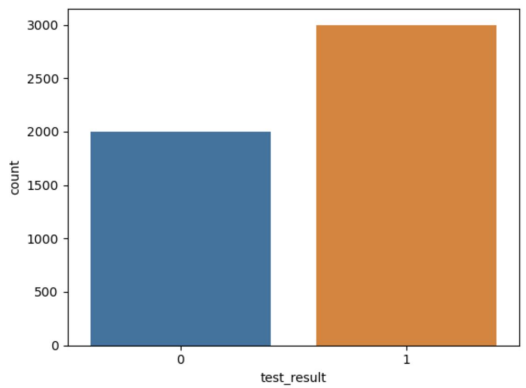
Out[49]:

	age	physical_score	test_result
0	33.0	40.7	1
1	50.0	37.2	1
2	52.0	24.7	0
3	56.0	31.0	0
4	35.0	42.9	1

Let's see how many people passed the hearing test and how many not:

```
In [50]: sns.countplot(data=df, x=df['test_result'])
         plt.show
```

```
Out[50]: <function matplotlib.pyplot.show(close=None, block=None)>
```

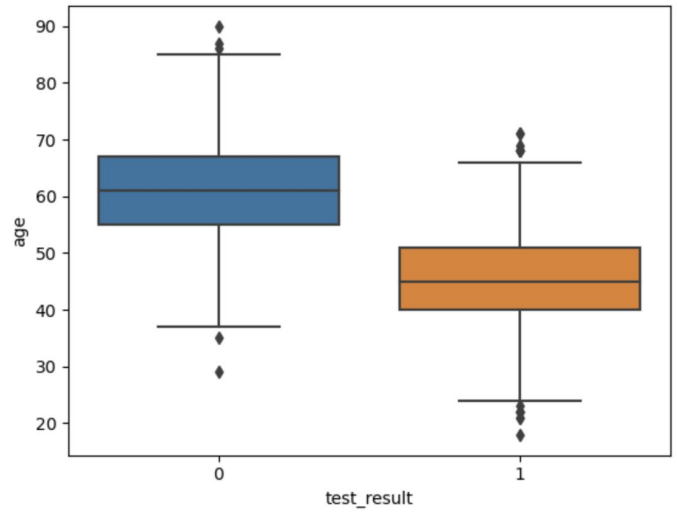


We can see that more people passed the test then won't.
We can also see that the amount of people passing and not passing is similar

Logistic Regression - Python Example

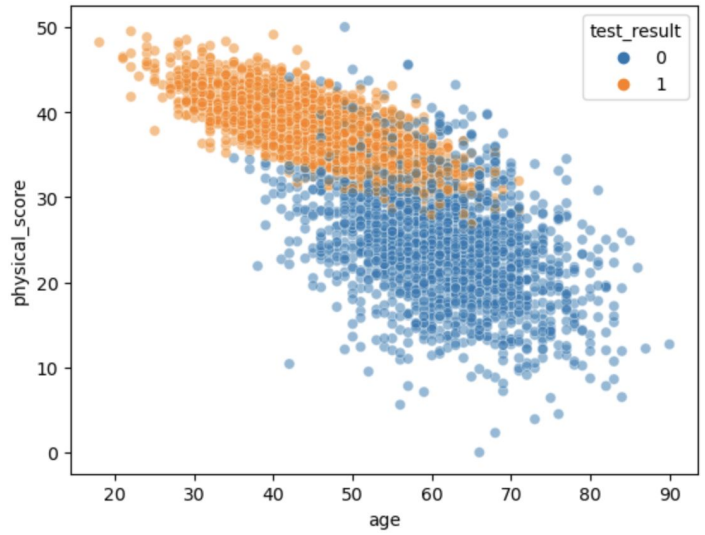
Let’s explore the correlation between the features and the test result:

```
In [51]: sns.boxplot(data=df, x='test_result', y='age')
         plt.show()
```



We can see that younger aged people are more likely to successfully pass the test then older age people

```
In [52]: sns.scatterplot(data=df, x='age', y='physical_score', hue='test_result', alpha=0.5)
         plt.show()
```



We can see that younger people tend to get higher physical score and people with high physical score are more likely to successfully pass the test.

Logistic Regression - Python Example

Now, let's perform logistic regression on our data in order to predict future test results.

For that we need to import the LogisticRegression() model and perform some data processing.

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

X = df.drop('test_result', axis=1)
y = df['test_result']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=101)

scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)


log_model = LogisticRegression()
log_model.fit(scaled_X_train, y_train)
```

Out[10]:
```
▾ LogisticRegression
LogisticRegression()
```

Same with linear regression, we want to perform train / test split

We also want to perform data processing, in this case standardization

We train our LogisticRegression() model on our scaled train dataset

Logistic Regression - Python Example

Once we train our model, we can see the model beta coefficient values and predictions:

```
In [56]: log_model.coef_
Out[56]: array([[-0.94953524,  3.45991194]])
```

We got positive coefficient for the ‘physical score’ feature, meaning the higher the physical score the more likelihood to get 1 result (pass the test)

We got negative coefficient for the ‘age’ feature, meaning the higher the age the less likelihood to get 1 result (pass the test)

```
In [57]: y_predictions = log_model.predict(scaled_X_test)
         y_predictions

Out[57]: array([1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1,
                0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
                0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
                0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
                1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
                1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
                1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
                1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
                1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
                1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
                0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
                1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
                0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0,
                1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0])
```

Printing the model predictions on the test dataset will give us values of 0 and 1 which indicate the model predictions of the person to pass or not

Logistic Regression - Python Example

We can also use ‘predict_proba()’ to see the predictions probability of getting each category:

```
In [60]: y_predictions = log_model.predict_proba(scaled_X_test)
         y_predictions

Out[60]: array([[2.38434328e-02, 9.76156567e-01],
                [2.69240761e-02, 9.73075924e-01],
                [9.89194168e-01, 1.08058325e-02],
                [1.90768955e-03, 9.98092310e-01],
                [9.75012619e-01, 2.49873806e-02],
                [9.89652504e-01, 1.03474957e-02],
                [7.40226674e-02, 9.25977333e-01],
                [1.70943342e-02, 9.82905666e-01],
                [9.97066033e-01, 2.93396692e-03],
                [3.30521615e-02, 9.66947839e-01],
                [8.31035903e-02, 9.16896410e-01],
                [9.92865306e-03, 9.90071347e-01],
                [7.08965218e-03, 9.92910348e-01],
                [9.34236379e-01, 6.57636209e-02],
                [1.16594927e-04, 9.99883405e-01],
                [6.58904151e-02, 9.34109585e-01],
                [9.89126706e-01, 1.08732938e-02],
                [2.80051357e-03, 9.97199486e-01],
                [5.58920624e-04, 9.99441079e-01],
```

The first value is the probability of getting 0 and the second value is the probability of getting 1 (Which is 1 - P(0))

Note: Our model default threshold for deciding what category it should predict is 0.5.

Meaning all prediction probabilities that are equal or above 0.5 will get 1 (success) prediction value and all prediction probabilities that are lower than 0.5 will get 0 (fail) prediction value.

Classification Metrics - Introduction

Just like in linear regression, we also have metrics that will help us evaluate our logistic regression model performance, but because we are predicting binary categorical data those metrics are different than the continuous data.

When dealing with classification models we need to use classification metrics.

Our model has 4 possible outcomes regarding it's prediction:

1. Unsuccess result predicted as unsuccess (0 - real result, 0 - predicted)
2. Unsuccessful result predicted as success (0 - real result, 1 - predicted)
3. Successed result predicted as success (1 - real result, 1 - predicted)
4. Successed result predicted as unsuccess (1 - real result, 0 - predicted)

Classification Metrics - Confusion Matrix

To better understand this let's take the following example →

Imagine that we developed a test that will help us detect if a person got infected with a virus based on some biological features.

In total we have a test dataset of 100 persons and our model predict 94 of the persons to be virus negative and 6 persons to be virus positive

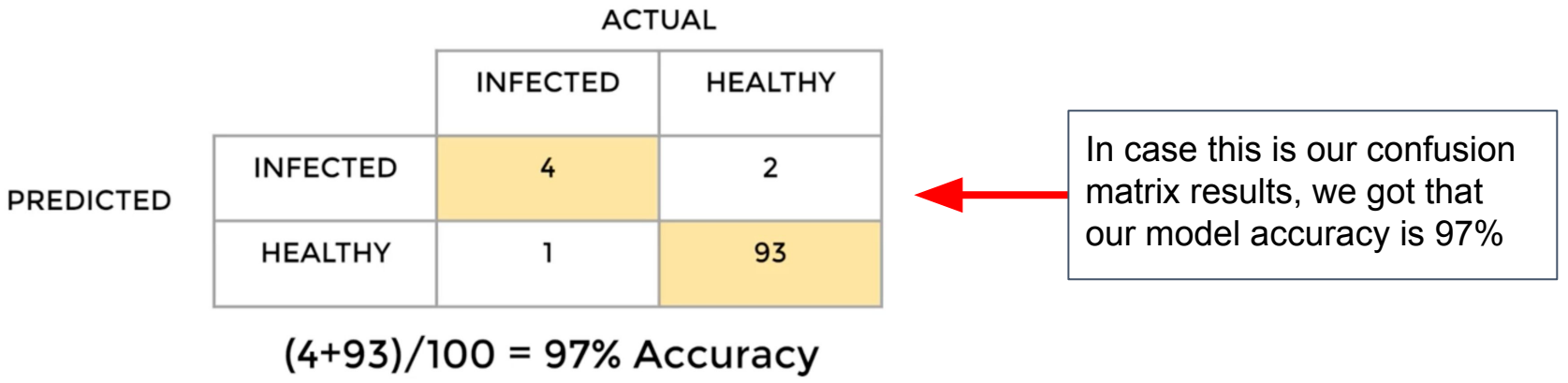
So based on the 4 possible outcomes of our model prediction we can build the following matrix which called **confusion matrix**:

		ACTUAL	
		INFECTED	HEALTHY
PREDICTED	INFECTED	TRUE POSITIVE	FALSE POSITIVE
	HEALTHY	FALSE NEGATIVE	TRUE NEGATIVE

Classification Metrics - Accuracy

Accuracy metric → The accuracy metric is the fraction of predictions that a classification model got right. Mathematically, it is defined as the number of correct predictions (both positive and negative divided by the total number of predictions made.

Accuracy = (True Positives + True Negatives) / (True Positives + True Negatives + False Positives + False Negatives)



Classification Metrics - Accuracy Paradox

The accuracy paradox refers to a situation in machine learning where a model's accuracy can appear to be misleadingly high despite the model actually having little predictive value. This typically occurs with imbalanced datasets, where one class of data greatly outnumbers the other class.

For example → Let's say our virus test model only predict negative and never predict positive.
We know that this model is not a good model and will most likely fail with real data but let's calculate this model accuracy:

		ACTUAL	
		INFECTED	HEALTHY
PREDICTED	INFECTED	0	0
	HEALTHY	5	95

(0+95)/100 = 95% Accuracy

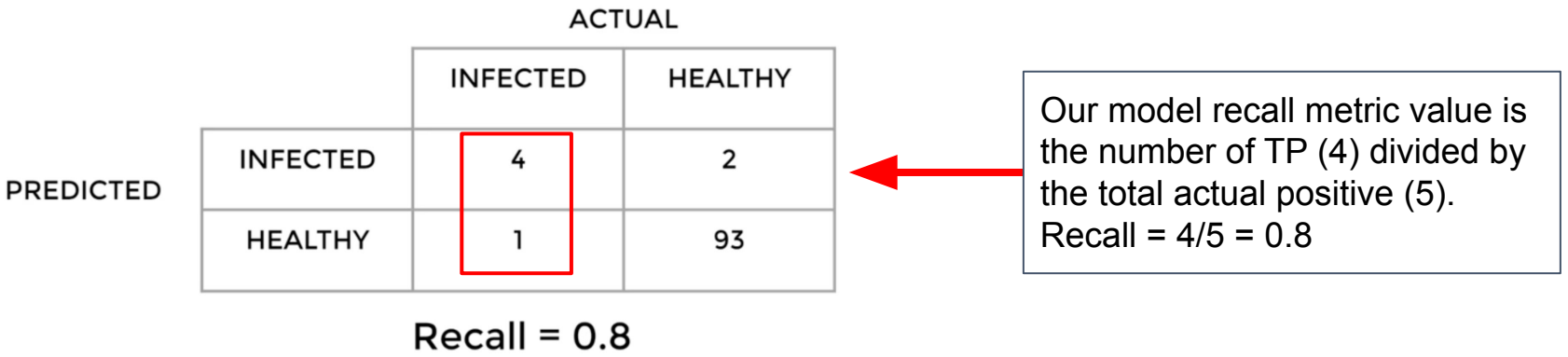
We got that our 'only negative' model accuracy is 95% percent despite the fact that it predict only one outcome no matter the data

Classification Metrics - Recall

In order to deal with the accuracy paradox we have 3 more metrics that can help us better evaluate our model.

Recall metric → Recall, also known as sensitivity, hit rate, or true positive rate (TPR), is a metric used in machine learning to measure the completeness or the effectiveness of a classifier in identifying positive instances.

Recall = True Positives / (True Positives + False Negatives)



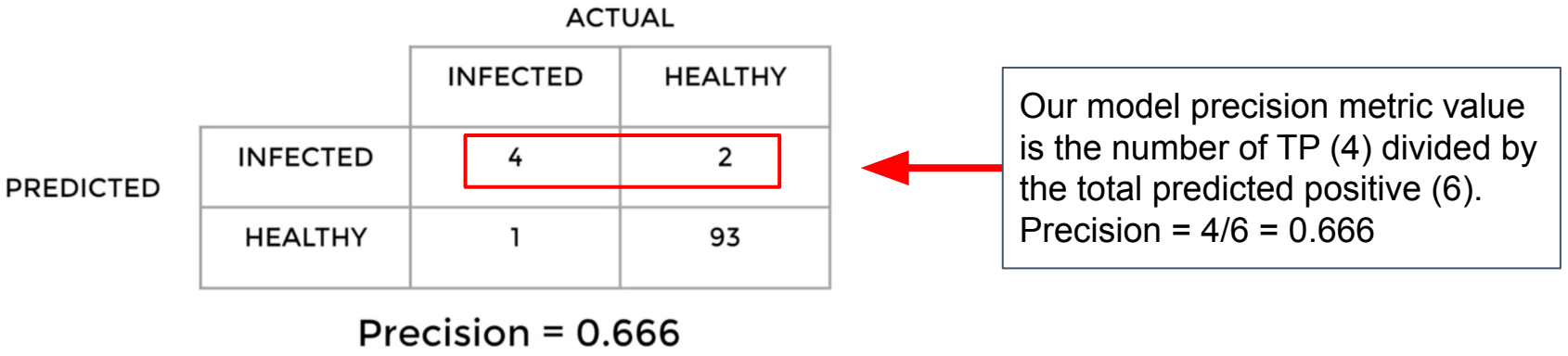
In our ‘always negative’ model example the recall value will be 0 (0 / 5 = 0).

When getting a recall value of 0 it should alert us that our model not catching any relevant cases.

Classification Metrics - Precision

Precision metric → Precision, also known as the positive predictive value, is a metric used in machine learning to measure the relevance or the correctness of the results obtained. It is particularly useful in situations where the cost of a false positive is high.

Precision = True Positives / (True Positives + False Positives)



In our ‘always negative’ model example the precision value will be 0 (0 / 0 = 0).

When getting a precision value of undefined it should alert us that something is not right with our model and we most likely failing to the accuracy paradox.

Classification Metrics - F1 Score

F1 Score metric → The F1 Score is a measure of a model's performance in the field of machine learning. It is a type of weighted average, or harmonic mean, of precision and recall. An F1 score can be considered a combination of these two measurements.

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

If either Precision or Recall is zero, the denominator in the F1 score calculation becomes zero, thus making the overall F1 score zero. This highlights that even if one of Precision or Recall is performing well but the other is zero, the model isn't effective.

In our example the F1 Score of our model will be -
F1 Score = (2 * 4/5 * 4/6) / (4/5 + 4/6) = 0.72

Classification Metrics - Python Example

Once we have our model predictions on the test set we can check the different classification metrics to check if our model is accurate and balanced.

```
In [19]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

         y_pred = log_model.predict(scaled_X_test)

         accuracy_score(y_test, y_pred)

Out[19]: 0.93
```

Our model accuracy is 0.93 meaning 93% of our model predictions are the correct predictions

We can also see the confusion matrix of the model:

```
In [20]: confusion_matrix(y_test, y_pred)

Out[20]: array([[172,  21],
                [ 14, 293]])
```

We have the following results:
172 - TP, 21 - FP
14 - FN, 293 - TN

Classification Metrics - Python Example

We can also plot a heatmap (using seaborn.heatmap() method) that will show the confusion matrix as a plot so it will be more readable:

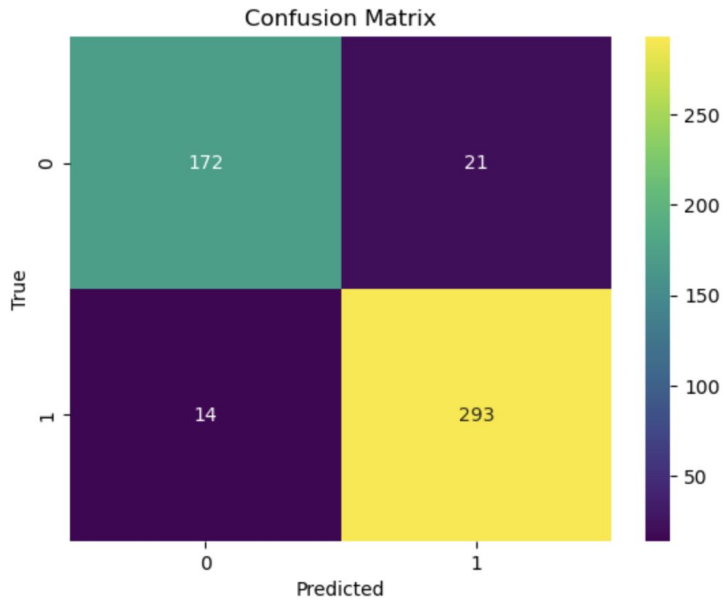
```
In [21]: cm = confusion_matrix(y_test, y_pred)

         cmap = plt.cm.viridis

         sns.heatmap(cm, annot=True, fmt="d", cmap=cmap)

         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('True')

         plt.show()
```



We are passing to the heatmap() method the following parameters:
cm → The confusion matrix, the heatmap data source.
annot → Determine if to write the data value in each cell.
fmt → Determine the data formatting code (“d” means base-10 integer)
cmap → determine the colors for the heatmap

The plot result is the same confusion matrix from previously but plotting in a more readable way

Classification Metrics - Python Example

In order to see the **recall** and **f1-score** of our model we need to execute the following:

```
In [27]: from sklearn.metrics import precision_score, recall_score

         recall_score(y_test, y_pred)

Out[27]: 0.9543973941368078
```

Our model recall metric is 0.954

```
In [28]: from sklearn.metrics import precision_score, recall_score

         precision = precision_score(y_test, y_pred)
         recall = recall_score(y_test, y_pred)

         f1_score = 2 * (precision * recall) / (precision + recall)
         f1_score

Out[28]: 0.9436392914653785
```

Once we calculated the precision and recall metrics values we can also calculate the f1-score value using the formula.

Our model f1-score metric is 0.943

Classification Metrics - Python Example

We can also generate classification_report that will provide all classification metrics (precision, recall, f1-score) on each class separately:

```
In [29]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.89	0.91	193
1	0.93	0.95	0.94	307
accuracy			0.93	500
macro avg	0.93	0.92	0.93	500
weighted avg	0.93	0.93	0.93	500

Using the classification_report we can see for the 0 class and for the 1 class all the metrics values (precision, recall and f1-score)

Class Exercise - Logistic Regression

Instructions:

Use the **‘titanic’** dataset provided by Seaborn library, Your mission is to predict if future passengers will likely to survive the titanic or not.

perform Logistic Regression machine learning modeling to with the following instructions:

- Get the ‘titanic’ data set by running the following command:

  titanic = sns.load_dataset("titanic")
- Create a new dataset continuing only the following columns: **‘age’, ‘sex’, ‘fare’, ‘survived’**

  The new dataset will provide data about each passenger in the titanic, the data will contain the passenger age, sex and the ticket price. In addition the ‘survived’ column determine if the passenger managed to survive or not.
- Apply data processing and remove rows with ‘null’ values in any column
- Change the ‘sex’ string values to numeric values so the machine learning model will be able to use the data. For example ‘male’ will be 0 and ‘female’ will be 1. (hint → use pd.get_dummies())

Class Exercise - Logistic Regression

Instructions:

- Apply simple Logistic Regression to predict likelihood of survivor
- Print your model prediction, Accuracy, Precision, Recall and F1-score metrics
- Plot your model confusion matrix
- In case your model is not accurate enough, think about ways to improve it (no need to change the code but just think about actions that could improve your model prediction)

Class Exercise Solution - Logistic Regression