**Titanic Survival with Machine Learning**

In this project, I will take you through a very famous case study for machine learning practitioners which is to predict titanic survival with Machine Learning. I will first introduce you to this case study and then I will show you how we can build a predictive model to predict survival with Machine Learning.

## **Machine Learning Case Study: Titanic Survival Analysis**

The sinking of the Titanic is one of the most infamous wrecks in history. On April 15, 1912, during her maiden voyage, the RMS Titanic, widely considered “unsinkable”, sank after hitting an iceberg.

Unfortunately, there were not enough lifeboats for everyone on board, resulting in the deaths of 1,502 out of 2,224 passengers and crew. While there was an element of luck in survival, it appears that certain groups of people were more likely to survive than others.

Here, your challenge is to build a predictive model that can give a solution to the question, “What types of people were more likely to survive?” using passenger data (i.e. name, age, sex, socio-economic class, etc.).

## Predict Titanic Survival with Machine Learning

Now, as a solution to the above case study for predicting titanic survival with machine learning, I’m using a now classic data set, which relates to passenger survival rates on the Titanic, which sank in 1912. I’ll start this task by loading the test and training dataset using pandas:

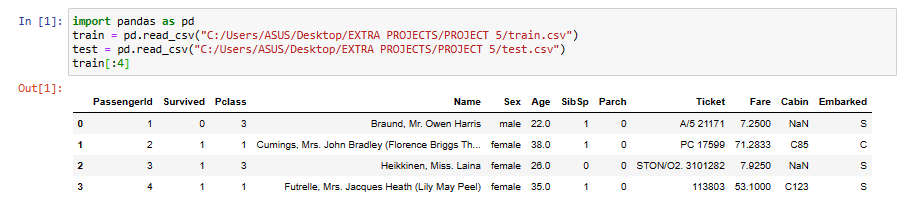


Fig 1

Scikit-learn’s algorithms generally cannot be powered by missing data, so I’ll be looking at the columns to see if there are any that contain missing data:

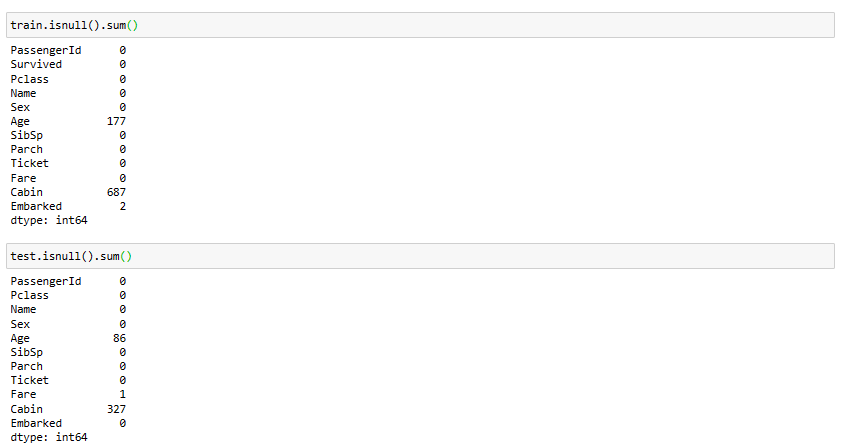


Fig 2

## Data Preparation

In statistics and machine learning examples like this, a typical task is to predict whether a passenger will survive based on the characteristics of the data. A model is fitted to a training data set and then evaluated on an out-of-sample test data set.

I would like to use Age as a predictor, but data is missing. There are several ways to do missing data imputation, but I’ll make a simple one and use the median of the training dataset to fill in the null values ​​in both tables:

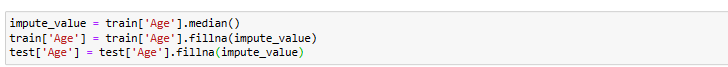


Fig 3

We now need to specify our models. I’ll add an Is Female column as the encoded version of the ‘Sex’ column:



Fig 4

Next, we decide on some model variables and create NumPy arrays:

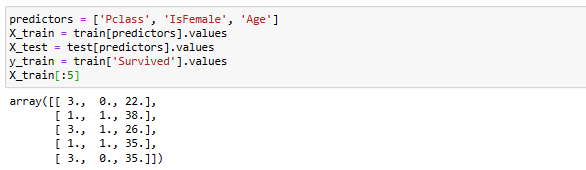


Fig 5

Now I’m going to use the LogisticRegression model from scikit-learn and create a model instance:



Fig 6

Now we can fit this model to the training data using the scikit-learn’s fit method:



Fig 7

Now, we can make predictions on the test dataset using model.predict:

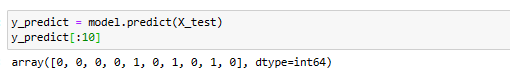


Fig 8

In practice, there are often many additional layers of complexity in training the models. Many models have parameters that can be adjusted, and there are techniques such as cross-validation that can be used for parameter tuning to prevent overfitting of training data. This can often improve predictive performance or the robustness of new data.

## Implementing Cross-Validation

Cross-validation works by splitting training data to simulate out-of-sample prediction. Based on a model accuracy score such as the root mean square error, one can perform a grid search on the model parameters. Some models, like logistic regression, have classes of estimators with built-in cross-validation.

For example, the Logistic Regression CV class can be used with a parameter indicating the degree of precision of a grid search to be performed on the model regularization parameter C:

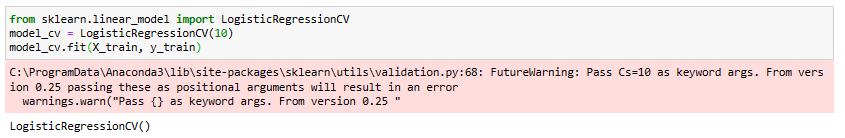


Fig 9

To perform cross-validation by hand, you can use the cross\_val\_score helper function, which handles the process of splitting data. For example, to validate our model with four non-overlapping divisions of training data, we can do:

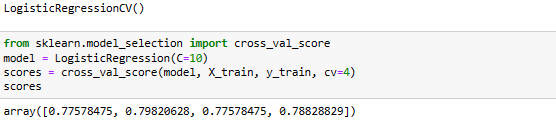


Fig 10

The default rating metric depends on the model, but it is possible to choose an explicit rating function. Cross-validated models take longer to train, but can often improve model performance.