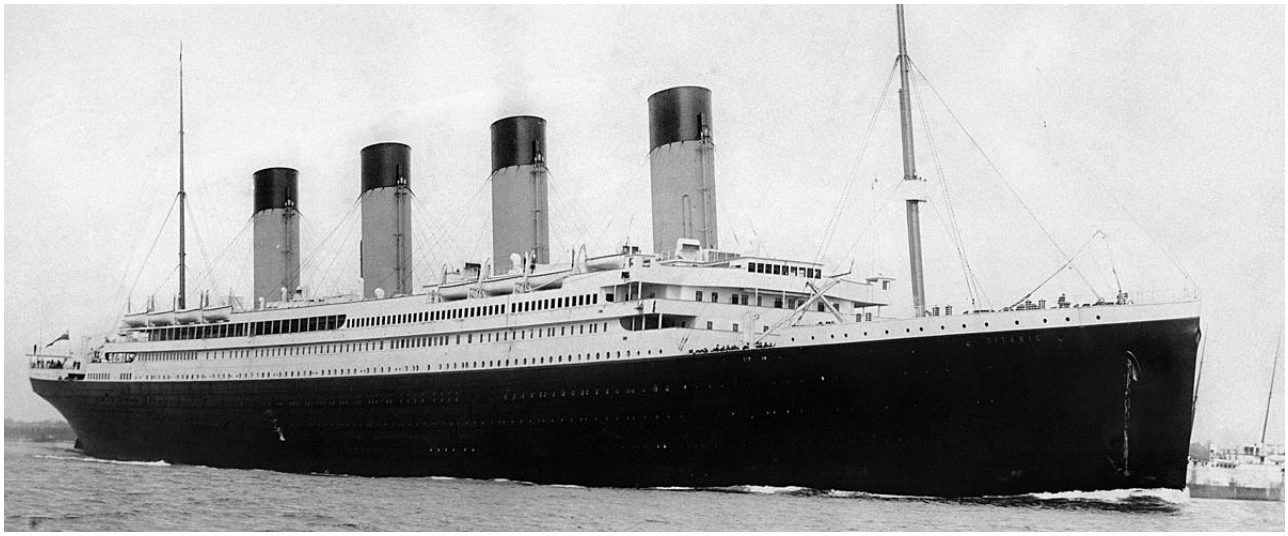
Machine Learning Engineer Nanodegree

Capstone Project
Titanic: Machine Learning from Disaster

Kevin Gonzalez – August 20, 2020



Definition

Project Overview

> *"The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered "unsinkable" RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc)." - Kaggle*

In this project I tackle the very interesting problem of making predictions with Machine Learning, in this case I focus on Binary Classification in relation to the above requirement as stated by Kaggle.

Predicting future events is a very real use case of Machine Learning, one that is quite applicable in this modern world of artificial intelligence.

The data sets and more information can be found on the associated Kaggle challenge webpage. Below is a summary of the data in the form of a data dictionary, provided by Kaggle

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Problem Statement

Given the two provided data sets (train and test, test having no ground truth labels), our goal is to create a Binary Classifier capable of making accurate predictions on the given test set.

"use the Titanic passenger data (name, age, price of ticket, etc) to try to predict who will survive and who will die." – Alexis Cook (Kaggle)

Some key characteristics of our data include age and gender. I've plotted these points below (y=number of survivors, x=metric, green=survivor, grey=non-survivor)

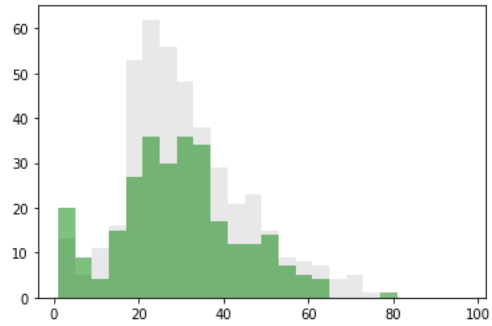


Figure 1: Age

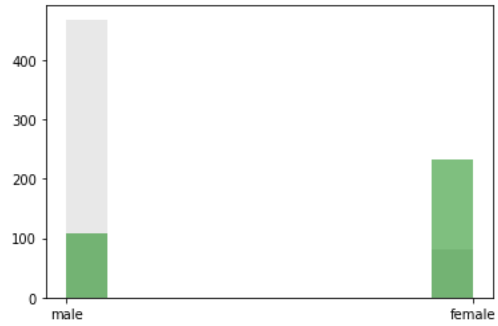


Figure 2: Gender

These data points and others show some obvious correlations between surviving and a given passengers characteristics, so the problem is definitely possible to solve.

However, we have neither many features nor a large sum of data. Therefore, I will be moving forward with XGBoost to tackle this problem. I will also take advantage of SageMakers hyperparameter tuning to create the most able XGBoost model.

Metrics

We will be using two metrics to benchmark and evaluate our model, the first is a validation data set created from our train data set. This will be used to validate our XGBoost model before making a submission to Kaggle.

The second will be the accuracy score provided by Kaggle, which we will receive after making a submission.

In both cases, accuracy is being used as the evaluation metric.

$$accuracy = (tp + tn) / (tp + fp + tn + fn)$$

Analysis

Data Exploration

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Table 1: Train dataset preview

Previewing the train data set a few things stand out. We can already see that the data is organized very well, and there are some obvious features we can work with, such as `Age`, `Sex`, `Pclass` (ticket class), `sibsp` (# of siblings / spouses aboard the Titanic), and `parch` (# of parents / children aboard the Titanic).

Other features like `Name`, prove to be less important.

We also see our `Classifier` labeled here as `Survived`. This is obviously missing from the test set.

From our initial observations, we can also see that we might be able to engineer some new features from this data, for instance; we can create a `Child` feature based on age, as well as an `Senior` feature.

Exploratory Visualization

We will primarily be visualizing the data using matplotlib.

Besides age and gender as visualized above, other features correlating to a passenger's survival include ticket class, number of siblings / spouses aboard the Titanic, number of parents / children aboard the Titanic, port of embarkation, and fare.

We can see this in the histograms below (again, y=number of survivors, x=metric, green=survivor, grey=non-survivor)

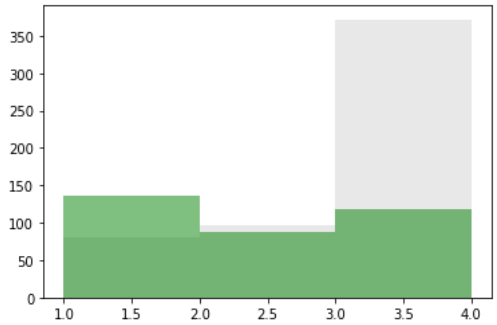


Figure 3: Ticket class

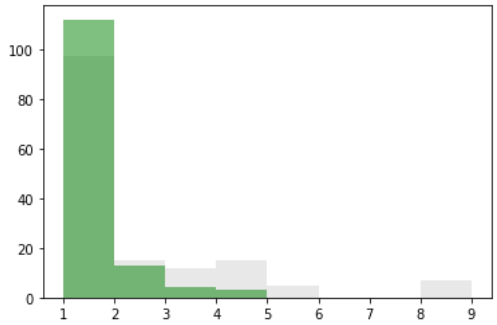


Figure 4: Number of siblings / spouses aboard the Titanic

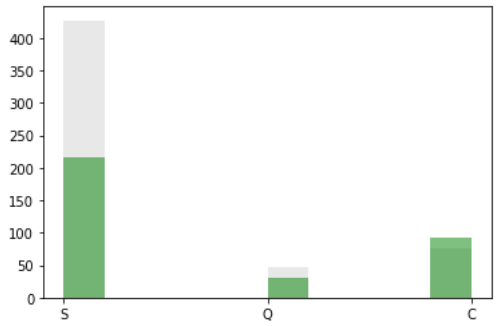


Figure 5: Port of embarkation

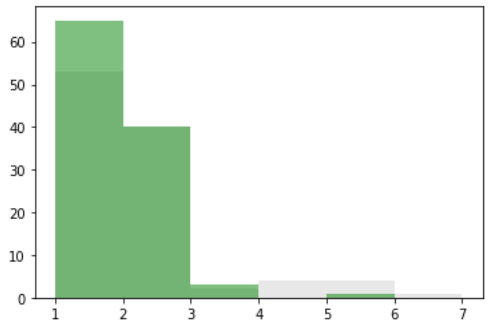


Figure 6: Number of parents / children aboard the Titanic

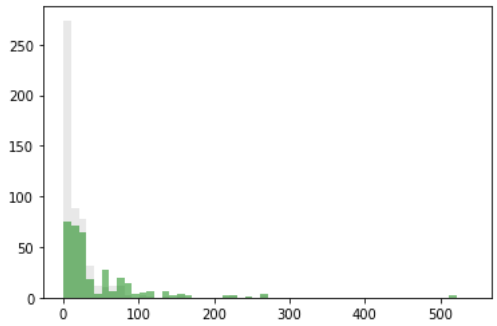


Figure 7: Fare

Although we didn't initial think `Embarked` or `Fare` would prove valuable, we do see some correlation, and I did choose to keep the features in the final model.

The histograms speak for themselves, these features all have some level of correlation, with ticket class and gender maybe being one of the two most crucial features.

Algorithms and Techniques

With the above features in mind, a binary logistic XGBoost model will surely fit this problem well, we need a model that can perform well even on a limited feature set with a moderate sample size. The model also needs to support binary classification. XGBoost should make for a fine choice.

In addition, we will take advantage of SageMaker's hyperparameter tuning to get the most out of our model.

Benchmark

Of course, we still need a benchmark to base our model's performance on, we will aim for a relatively high accuracy (calculated by our validation data and by Kaggle) of about 80% (give or take 5 – 10% for the Kaggle "Production" submission).

Methodology

Data Preprocessing

While organized, our data is far from perfect, for instance some features will not be used at all by our model, and a number of features can be normalized, these include Pclass, Age, SibSp, Parch, and Fare.

In addition, some data needs to be transformed into numerical values. Namely Embarked and Sex.

We've accomplished the first step of dropping unnecessary features by using our function `get_df_with_base_features`, this dropped all the features we won't be using from now on. The secibd step was accomplished using our `process_features` function, which maps Embarked and Sex to numerical values, and finally normalizes all of our numerical values.

With this function defined, we passed in our data frames and successfully processed our train and test data.

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Table 2: Stage one preprocessed train dataset preview

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0.0	1.0	1.0	0.271174	0.125	0.0	0.014151	1.0
1	1.0	0.0	0.0	0.472229	0.125	0.0	0.139136	0.0
2	1.0	1.0	0.0	0.321438	0.000	0.0	0.015469	1.0
3	1.0	0.0	0.0	0.434531	0.125	0.0	0.103644	1.0
4	0.0	1.0	1.0	0.434531	0.000	0.0	0.015713	1.0

Table 3: Stage two preprocessed train dataset preview

With that, we have our selected features, processed for our model.

Implementation

The first thing we are going to do is organize our data for our model. To do this, we create our "complete" data frames. The first one is easy, `complete_test_df ` is simply a copy of our `processed_test_df`, the test variant of our `processed_train_df ` visualized above.

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1.0	1.0	0.452723	0.000	0.000000	0.015282	0.5
1	1.0	0.0	0.617566	0.125	0.000000	0.013663	1.0
2	0.5	1.0	0.815377	0.000	0.000000	0.018909	0.5
3	1.0	1.0	0.353818	0.000	0.000000	0.016908	1.0
4	1.0	0.0	0.287881	0.125	0.111111	0.023984	1.0

Table 4: Preprocessed test dataset preview

The next two data frames are a little more complicated, but not too bad. Using `sklearn.model_selection.train_test_split` we split our `processed_train_df ` into two data frames, `complete_train_df` and `complete_val_df `, as indicated by the names, these two data frames represent our training and validation data sets.

We then upload our data to a local directory, and in turn we upload to s3.

With that we can finally construct our model.

Using SageMaker, we import an XGBoost container, create an XGBoost estimator, and set our XGBoost hyperparameters.

```
xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type="ml.m4.xlarge",
                                    output_path="s3://{}/{}/output".format(session.default_bucket(), prefix),
                                    sagemaker_session=session)

xgb.set_hyperparameters(max_depth=8,
                        eta=0.4461468909040185,
                        gamma=0.0019526615254589956,
                        min_child_weight=3,
                        subsample=0.8225794003726403,
                        silent=0,
                        objective="binary:logistic",
                        early_stopping_rounds=20,
                        num_round=500)
```

Figure 8: XGBoost model

We also need to setup our `HyperparameterTuner`, this way we can take full advantage of AWS and XGBoost.

```
from sagemaker.tuner import IntegerParameter, ContinuousParameter, HyperparameterTuner

xgb_hyperparameter_tuner = HyperparameterTuner(estimator = xgb,
                                               #objective_metric_name = "validation:rmse", reulted in accuracy of 0.78
                                               objective_metric_name = "validation:error", # resulted in an accuracy o
                                               objective_type = "Minimize",
                                               #max_jobs = 20, in test, the model stopped improving after 7 or 8 jobs
                                               max_jobs = 20,
                                               max_parallel_jobs = 3,
                                               hyperparameter_ranges = {
                                                    #"max_depth": IntegerParameter(3, 12), in test, higher depth perfo
                                                    "max_depth": IntegerParameter(3, 15),
                                                    "eta"      : ContinuousParameter(0.05, 0.5),
                                                    #"min_child_weight": IntegerParameter(2, 8), in test, higher min ch
                                                    "min_child_weight": IntegerParameter(2, 12),
                                                    "subsample": ContinuousParameter(0.5, 0.9),
                                                    "gamma": ContinuousParameter(0, 10)
                                               })
```

Figure 9: XGBoost HyperparameterTuner

Many of these parameters were altered as a result of refinement, but the original values can be seen in the comments. For the default parameters, these were selected based on results from previous models.

With the model created and ready to be tuned and fit, I collected the s3 input for train and validation, and tuned the hyperparameters. After attaching our best resulting training job, I created a transformer and created predictions for our validation data set.

The initial accuracy was 0. 7830508474576271, this misses our benchmark of 80%.

Refinement

The refinement process was broken into 3 iterations.

In **iteration one**, we adjusted the objective metric, finding that `validation:error` produced the best results. Our initial metric being `rmse`, as pictured in Figure 9.

In **iteration two** we made many changes, altering most parameters, this proved to be very difficult, but did result in a higher accuracy. The highest performing parameters we were able to achieve in iteration two were as follows:

max_depth=8,
eta=0.4461468909040185,
gamma=0.0019526615254589956,
min_child_weight=3,
subsample=0.8225794003726403

Finally reaching an accuracy of 0.8576271186440678; thus, meeting our goal of 80% accuracy!

In **iteration three** we finally created our engineering features `Child` and `Senior`.

I accomplished this by defining a `eng_features` function which built the `Child` and `Senior` features from the `Age` feature. Prior to building the new features, we filled all the NaN `Age` values with the mean age of our feature. `Child` is defined as an age of 12 or less. `Senior` was defined as an age of 55 or higher, and later to an age of 62 and higher.

In addition, I have added a `Name` feature based on the original `Name` feature. This was inspired by other Kaggle submissions, including this one https://www.kaggle.com/mrisdal/exploring-survival-on-the-titanic

Results

Model Evaluation and Validation

After our refinement process, we can make a few adjustments to our model, and we can finally validate it.

First let’s look at our new best parameters:

```
xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type="ml.m4.xlarge",
                                    output_path="s3://{}/{}/output".format(session.default_bucket(), prefix),
                                    sagemaker_session=session)

xgb.set_hyperparameters(max_depth=12,
                        eta=0.0011339607806480502,
                        gamma=3.885438849556937,
                        min_child_weight=2,
                        subsample=0.8684547185657749,
                        silent=0,
                        objective="binary:logistic",
                        early_stopping_rounds=20,
                        num_round=500)
```

Figure 10: Final best hyperparameters

As we can see, they are a tad bit different from our previous best.

With these parameters, the final accuracy on our validation set was 0.8666666666666667

We can also finally consider the Kaggle scores, impressively, our peak score was 0.77 on Kaggle.

It makes sense that a higher max depth might make a difference, in all the experiments, max depth was always fairly high, in some models as high as 44. Eta is lower than expected, but this is of course not a surprise due to the nature of machine learning. Likewise, gamma is higher than expected.

Our final scores are 0.8666666666666667 for our own validation metrics and 0.77 for Kaggles validation. This meets and exceeds our goal.

Justification

With that we are done. Our model has met my expectations (and surpassed them!)

It is not reasonable to continue to try to get a higher score on Kaggle at this point, 0.77 is an acceptable score and is in line with what many others seem to score, likewise the data at hand cannot be perfectly predicted (hence the goal of ~80%).

Therefore, I am happy to conclude this challenge and accept my results as a success, having met my goals.