In Woo Park

ICS 635

May 10, 2022

Titanic - Machine Learning from Disaster

**1 Introduction**

The beginning of the semester started with a Kaggle competition, therefore, I thought it would be appropriate to end with a Kaggle competition as well. The problem I chose was Titanic - Machine Learning From Disaster. The competition uses machine learning to create a model that predicts which passengers survived the Titanic shipwreck.

**2 Introduce the dataset**

The Kaggle competition: [Titanic - Machine Learning From Disaster](https://www.kaggle.com/c/titanic/data?select=train.csv), provides three different csv files. They are: gender\_submission.csv, test.csv, and train.csv. The gender\_submission.csv is an example file showing you that you need to submit 417 predictions with passengerID and if they survived or not, denoted by 0’s or 1’s. The train.csv contains 891 passengers with attributes: ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']. The test.csv contains 417 passengers with the same attributes except [‘Survived’] as we have to predict that ourselves. Most of the attribute data are nominal with the exception of [‘Pclass’] which represents the deck level passengers lived on.

**3 Introduce the model**

Throughout my attempt at the Kaggle competition, I chose to use K-Fold cross validation and logistic regression. K-Fold was chosen as there isn’t a lot of training data to work with (891/2224 = 0.40% of passenger data). Logistic regression was chosen as we use it when the dependent variable or target is categorical (i.e., Predict whether passenger survived (1) or not (0).

**4 Specify features and pre-processing.**

The main algorithm will be logistic regression, therefore, it would be easier to clean up the data by changing values to integers instead of strings once both train.csv and test.csv files are decoded for pandas dataframe. We will call the train.csv dataframe as train\_df or training set, and the test.csv dataframe as test\_df or test set. As mentioned before, both datasets contain the features ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'].

We know that the difference between the training dataset and the test dataset are the features provided. Namely, the test dataset is missing the "survived" attribute because we have to make a prediction if they survived. However, several attributes have missing values. These attributes are Age, Cabin, and Embarked for the training dataset. Embarked values are missing for 2 passengers only, namely, PassengerID: 62 and PassengerID: 830 in the training set. It seemed that the missing data was insignificant as the test dataset did not have missing Embarked values but missing values for Age, Cabin, and Fare. According to the challenge [overview](https://www.kaggle.com/competitions/titanic/overview), there were 1502 fatalities out of 2224 passengers and crew. This suggests our training data accounts for approximately (891/2224) = 0.40062.. or 40% of the actual number of passengers. Compared to the actual survival rate of, (2224-1502)/2224 = 0.3246.. or 32%, the training set has a 38% survival rate. For cabin data in the training set, 77% are NULL, 22% Other. For Embarked data in the training set, 72% S, 19% C, 9% Other. With this information in mind, we begin our attempt at cleaning the data.

To begin with, we check if any of the features are correlated with survival.

**Examples of Attributes Correlated with Survival**

|  | Pclass | Survived |
| --- | --- | --- |
| 0 | 1 | 0.63 |
| 1 | 2 | 0.47 |
| 2 | 3 | 0.24 |

|  | Sex | Survived |
| --- | --- | --- |
| 0 | female | 0.74 |
| 1 | male | 0.19 |

|  | SibSp | Survived |
| --- | --- | --- |
| 1 | 1 | 0.54 |
| 2 | 2 | 0.46 |
| 0 | 0 | 0.35 |
| 3 | 3 | 0.25 |
| 4 | 4 | 0.17 |
| 5 | 5 | 0 |
| 6 | 8 | 0 |

|  | Embarked | Survived |
| --- | --- | --- |
| 0 | C | 0.55 |
| 1 | Q | 0.39 |
| 2 | S | 0.34 |

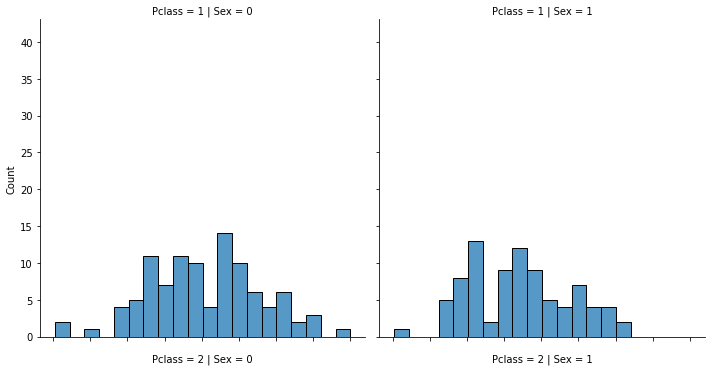
We notice that our correlation method doesn’t work for Ticket, Cabin, Fare, and Age. They are quantitative variables that don’t really help except Fare and Age. We will discuss later on how we change Fare and Age as integer ranges to help us during our preprocessing step. As for Ticket and Cabin, we drop those attributes (columns) from both datasets because when we review the data, there are either too many unique identifiers (unique values) for tickets in the training set and test set, and 686 missing values (NaN) for Cabin data in the training set (77% null) and 326 missing values for Cabin data in the test set (78% null).

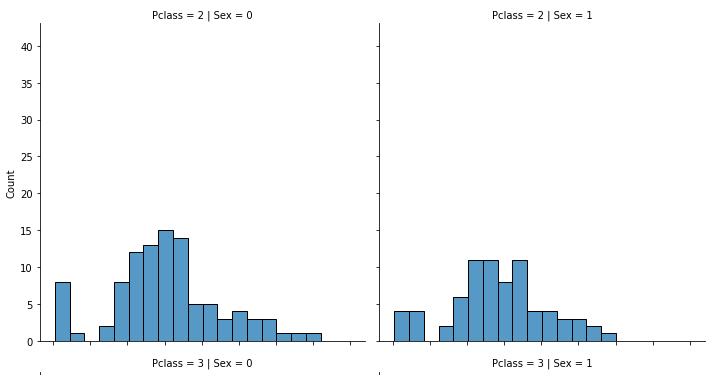
Next we replace Embarked values, which are denoted by (S, C, Q) to represent city names, into integer values (S → int(1), C → int(2), Q → int(3)). Two Embarked values in the training set are NaN. I decided to replace the two NaN values with int(1) as city “S” represents the majority at 72%, C at 19%, and Q at 9% in the training set.

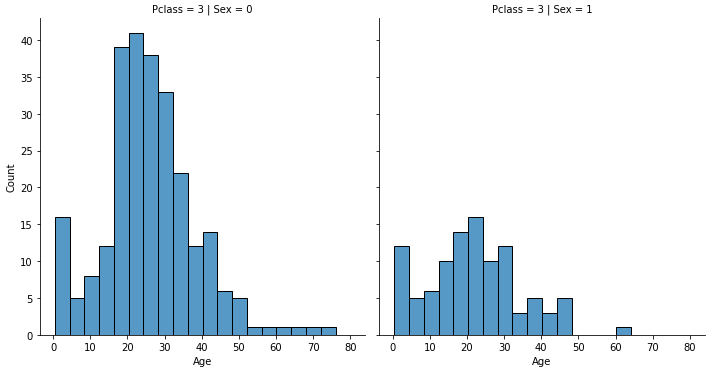
Similarly for Sex, we replace the strings “female” to int(1) and “male” to int(2). Sex attributes are not missing in either training or test datasets.

Next we attempt to clean up Names. At first, it seemed like Names weren’t helpful as it was similar to Cabin data where there were too many unique identifiers (unique names). However, after closer inspection, the Name attribute followed a [LastName, Prefix, FirstName] format. Therefore, we can change the Name attribute to categorical as well by dropping both last name and first name in both datasets, only keeping the Prefix (i.e., Mr, Miss, Mrs, Master, Other). The Prefix’s present are: ['Mr' 'Mrs' 'Miss' 'Master' 'Don' 'Rev' 'Dr' 'Mme' 'Ms' 'Major' 'Lady' 'Sir' 'Mlle' 'Col' 'Capt' 'Countess' 'Jonkheer']. We replace Mme → Mrs, Ms to → Miss, and Mlle → Miss. Finally, we change the values to integer values again for Mr → int(1), Miss → int(2), Mrs → int(3), Master → int(4), Other → int(5). We can now drop the Name attribute as we can use Prefix instead.

For Age, there are 177 missing values in the training set, and 86 missing values in the test set. We know that the Prefix, “Master” represents a person that is underage (< 18 years old), therefore, if Age values are missing but they have the “Master” prefix, we should give them a random value between 0 and 18. For the training set, only 4 Age values are NaN with Master, and for the test set, only 4 Age values are NaN with Master. To handle the remaining missing values for both data sets, we should calculate the standard deviation of Ages based on Pclass and Sex. The graph below is a visual representation of the standard deviation of Ages.







We use the numpy where() function on Pclass and groupby() function on Pclass over Age to find the standard deviation and mean of Age’s for each deck level. For example: print(df.where(df.Pclass == 1).groupby(['Pclass'])['Age'].describe())

which returns:

| **Deck Level** | **Mean** | **Standard Deviation** |
| --- | --- | --- |
| Pclass 1 | 38.23 | 14.80 |
| Pclass 2 | 29.88 | 14.00 |
| Pclass 3 | 24.96 | 12.53 |

This information allows us to predict missing values for Age for both data sets. Therefore, we set three variables as np.random.normal(mean, std), according to their respective mean and standard deviation values. We then go through both datasets and check if Age value is NaN and which Pclass they belong to, then assign a predicted Age value. We then categorize Age values into ranges based on pandas cut() function with 5 bins and find the likelihood of survival based on these Age ranges.

train\_df['Age\_Range'] = pd.cut(train\_df['Age'], 5)

train\_df[['Age\_Range', 'Survived']].groupby(['Age\_Range'], as\_index=False).mean().sort\_values(by='Age\_Range', ascending=True)

Which returns:

|  | Age\_Range | Survived |
| --- | --- | --- |
| 0 | (0.34, 16.336] | 0.52 |
| 1 | (16.336, 32.252] | 0.33 |
| 2 | (32.252, 48.168] | 0.40 |
| 3 | (48.168, 64.084] | 0.43 |
| 4 | (64.084, 80.0] | 0.09 |

We then continue our previous changes to the data by representing each Age\_Range as integer values (i.e. if (train\_df.Age[x] <= 16) → train\_df.Age[x] = int(0) ). This process is done to both datasets.

Finally, for Fare, the training set has 0 NaN values but the test set has one value NaN. We give the test set NaN value as int(0) instead and repeat the same process of binning Fare into Fare\_Ranges. However, this time we went with qcut because it creates unequal size bins but frequency of samples are equal in each bin. Fare values are not as easy to work with in this case because although Ages can span between 0.34 ←→ 80, Fare values are between 0 ←→ 512. Therefore, we would prefer to see bins spaced on the same frequency with unequal spacing than the other. Which returns:

|  | Fare\_Range | Survived |
| --- | --- | --- |
| 0 | (-0.001, 7.91] | 0.2 |
| 1 | (7.91, 14.45] | 0.30 |
| 2 | (14.454, 31.0] | 0.45 |
| 3 | (31.0, 512.329] | 0.58 |

We do the same process for Fare by assigning these ranges integer values. We finish our data processing by changing all attribute data types to integer values as some of them were floats.

**5 Specify data splits and how they are used**

We don’t have that much data for the training set, (only 891 passengers out of 2224 total passengers). This makes the choice of maximizing train/test/split difficult when choosing a split ratio, considering that we want the best validation for the test set as well. That is why we went with k-fold cross validation. This way we can split the training set (i.e. 5 splits), and we can use each unique group as a hold out or test set, and the remaining groups as training sets, a notable solution for our train/test/split maximization problem. Each group is given an opportunity to be used in the hold out set one time and also used to train the model k-1 times. This should show if we are overfitting or underfitting as we train on k data splits, which can provide insight on which training folds perform well compared to other training folds. I attempted this with 5 splits in the final submission but my previous choice was 3 splits as it was easier to follow,

(i.e., Model 1: Train on Fold1+Fold2, Test on Fold3,

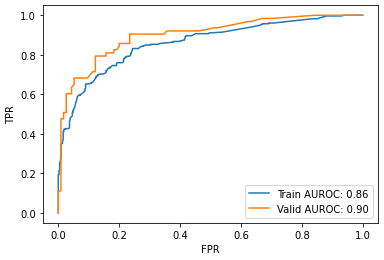
Model 2: Train on Fold2+Fold3, Test on Fold1,

Model 3: Train on Fold1+Fold3, Test on Fold2).

The results of using kfold and fitting to logistic regression is provided below:

| Fold # | Accuracy (train) | Accuracy (valid) | AUROC (train) | AUROC (valid) |
| --- | --- | --- | --- | --- |
| Fold 1 | 80.20% | 79.89% | 0.87 | 0.84 |
| Fold 2 | 80.50% | 78.65% | 0.87 | 0.86 |
| Fold 3 | 80.22% | 80.34% | 0.87 | 0.87 |
| Fold 4 | 81.21% | 75.28% | 0.87 | 0.85 |
| Fold 5 | 79.38% | 84.27% | 0.86 | 0.90 |

A graph of the results comparing True Positive Rate over False Positive Rate is provided below:

****

The results of the k-fold cross validation shows that on average, the model is slightly overfit. For example, the graph above illustrates the area under the receiver operating characteristic (AUROC) results from the final fold only. If we judged the model on this criteria alone, we should believe the model will perform similarly on the test data. However, when we look at the results across all folds, we can see 3 of 5 folds perform worse on the validation set when compared to the training set, when using accuracy and AUROC as a metric. In fact, across 5 folds, the validation set AUROC is about 0.864, still less than the average training set performance. Therefore, we should expect the final submission to achieve an accuracy of about 79% on the test set, or slightly less than this due to overfitting.

**6 Specify the hyperparameter search space & how hyperparameters were optimized**

Logistic regression is a robust algorithm, therefore, logistic regression does not really have any critical hyperparameters to tune other than perhaps observing differences in performances with different solvers or penalty regularization. Therefore we don’t do anything for hyperparameters. The sklearn logistic regression defaults to using "lbfgs" optimization and l2 regularization.

**7 Evaluate the model on a clean test set.**

Kaggle expects a file\_name.csv submission file with 418 rows with Passenger ID between 892 and 1309, and a subsequent survival prediction in the Survived column with 0 for dead, and 1 for survived. After we train with logistic regression, we attempt our predictions on the test set and download the file to submit to Kaggle. Once submitted, Kaggle returned a “score” value of 0.78229, where “score” is a percentage ratio of the correct number of predictions over the total number of predictions accuracy. Which means we correctly predicted 326/418 passengers on the test set.

**8 Explain any differences in the train/test datasets.**

This competition has existed since 2012, so I don’t believe there could have been any data drift. The only noticeable difference as of writing is the missing attribute of Survived in the test set which is intended.

**10. Conclusion**

Obviously, our model has a prediction accuracy on the test set of 78%. Observing our accuracy and AUROC values on each k splits, we can conclude that our expected prediction accuracy is at least between 75% - 80%. It’s understandable to question why we would consider this range but we should consider that accuracy is a simple way of measuring the effectiveness of our model, but it also can be misleading. Which is why we prefer to use AUROC values on train and validation instead. It would be interesting to observe the behavior of the model on another test set to confirm our expected prediction. All in all, I have placed #2498 on the Kaggle competition leaderboard, which could have potentially been greater if the top submissions were not clouded by cheated submissions. (People figured out a way to re-submit the example file to get a score of 1.0, or people used third party information revealing passenger survival data to accurately make a prediction).